Foundation

ASP for Dreamweaver 8

Omar Elbaga
Rob Turnbull

Set up your own testing environment including ASP, SQL Server, and Access

Master the Dreamweaver 8 ASP server behaviors

Construct real-world applications, including a photo gallery and a blog

friendsof ED
DESIGNER TO DESIGNER™
an Apress company

Foundation ASP for Dreamweaver 8

Omar Elbaga and Rob Turnbull

Foundation ASP for Dreamweaver 8

Copyright © 2006 by Omar Elbaga and Rob Turnbull

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 978-1-59059-568-8

ISBN (pbk): 1-59059-568-8

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editor
Chris Mills

Technical Reviewer
Jason Nadon

Editorial Board
Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser

Project Manager
Beth Christmas

Copy Edit Manager
Nicole LeClerc

Copy Editors
Damon Larson, Nicole LeClerc

Assistant Production Director
Kari Brooks-Copony

Production Editor
Ellie Fountain

Compositor
Dina Quan

Proofreader
Lori Bring

Indexer
John Collin

Artist
April Milne

Interior and Cover Designer
Kurt Krames

Manufacturing Director
Tom Debolski

I would like to dedicate this book to my mom; dad; brothers and sister, Tamer, Hesham, Soliman, and Yasmine; my fiancée, Mona; my entire family; and all my faithful readers.

—Omar Elbaga

This book is dedicated to my family and friends who always support me. Special thanks go to baby Theo who always puts an even bigger than usual smile on my face.

—Rob Turnbull

CONTENTS AT A GLANCE

About the Authors . . . . . **xiv**
About the Technical Reviewer . . . . . **xv**
Acknowledgments . . . . . **xvi**
Introduction . . . . . **xvii**

Chapter 1 **Dreamweaver and Dynamic Sites** **1**

Chapter 2 **The Dreamweaver Environment** **19**

Chapter 3 **A First Taste of ASP** **45**

Chapter 4 **Databases** **83**

Chapter 5 **Working with Forms** **133**

Chapter 6 **Building a Random Quote Generator** **175**

Chapter 7 **Completing the Quotes Administration System** **195**

Chapter 8 **Restricting Access** **223**

Chapter 9 **Creating a Blog** **245**

Chapter 10 **Image Gallery** **287**

Index **327**

CONTENTS

About the Authors . . . . . **xiv**
About the Technical Reviewer . . . . . **xv**
Acknowledgments . . . . . **xvi**
Introduction . . . . . **xvii**

Chapter 1 **Dreamweaver and Dynamic Sites** **1**

Data exchange 2
Dynamic takes over static 7
Dreamweaver server models 7
The ASP VBScript server model 7
What Dreamweaver 8 offers you as an ASP developer 9
Checking out data 9
Connecting to databases 9
Where is ASP in Dreamweaver? 10
ASP features supported by Dreamweaver 10
Insert bar 11
The Insert bar's ASP tab 12
The Insert bar's Application tab 14
Application panel 16
The Databases tab 16
The Bindings tab 16
Conclusion 17

Chapter 2 **The Dreamweaver Environment** **19**

Workspace layout 19
The Multiple Document Interface 20
The CSS panel 21
The Application panel group 22
The Databases panel 22
The Bindings panel 23
The Server Behaviors panel 24
The Tag Inspector panel group 25
The Attributes panel 26
The Behaviors panel 26

0e197eab08414336a67d4228a6088055

The Files panel group . . . 27
The Files panel . . . 27
The Assets panel . . . 28
The Snippets panel . . . 28
The Properties panel . . . 29
The Results panel group . . . 30
The Search panel . . . 30
The Validation panel . . . 30
The Target Browser Check panel . . . 31
The Link Checker panel . . . 31
The Site Reports panel . . . 31
The FTP Log panel . . . 31
The Server Debug panel . . . 32
The Reference panel . . . 32
Toolbars . . . 32
The Insert toolbar/panel group . . . 32
The Document toolbar . . . 33
The Standard toolbar . . . 35
The Style Rendering toolbar . . . 35
The Coding toolbar . . . 36
New tools in Design view . . . 37
Zoom . . . 37
Magnification . . . 37
Hand . . . 37
Guides . . . 37
Defining an ASP VBScript site in Dreamweaver . . . 39
Local Info screen . . . 39
Remote Info screen . . . 40
Testing Server screen . . . 41
Creating the site . . . 42
Conclusion . . . 43

Chapter 3 **A First Taste of ASP** . . . **45**

Specify your language: VBScript . . . 46
ASP delimiters <% %> . . . 46
Insensitive case! . . . 47
Variables . . . 48
Declaring and printing variables . . . 49
Letting VBScript declare variables implicitly . . . 52
Syntax rules for variables . . . 52
Data types: string, integer, and Boolean . . . 52
Variable concatenation . . . 54
Watching out for “adding” numbers . . . 55
Variable naming conventions . . . 56
Prefixes . . . 56
Using understandable variable names . . . 56
Commenting code . . . 57

VBScript's house of built-in functions . . . 58
Conversion functions . . . 58
String functions . . . 59
Operators . . . 61
Assignment operators . . . 62
Logical operators . . . 62
AND, OR . . . 62
NOT . . . 63
Comparison operators . . . 64
Mathematical operators . . . 65
Operator precedence . . . 66
Conditional logic . . . 68
The If statement . . . 68
The If . . . Then . . . Else statement . . . 69
The If . . . Then . . . ElseIf statement . . . 69
Looping logic . . . 71
The Do . . . Loop statement . . . 71
The While . . . Wend statement . . . 72
Cookies (ASP cookies, not chocolate chip!) . . . 72
Response.Cookies and Request.Cookies . . . 73
Cookie expiration . . . 75
Updating cookies . . . 75
Deleting cookies . . . 76
A word of warning about the use of cookies . . . 76
Session variables . . . 76
Setting the session timeout interval . . . 77
Creating and retrieving session variables . . . 77
Updating session variables . . . 78
Deleting session variables . . . 78
Note on sessions . . . 79
Environment variables . . . 79
Conclusion . . . 81

Chapter 4 **Databases** . . . **83**

Creating a database . . . 84
Inside Microsoft Access . . . 84
Inside Microsoft SQL Server . . . 85
Preparing a SQL Server database for Internet use . . . 88
Creating the IUSR account in SQL Server . . . 89
Database design . . . 91
Object naming conventions . . . 91
Creating tables . . . 92
Creating tables in Access . . . 93
Creating tables in SQL Server . . . 95
Relational databases and referential integrity . . . 97
Creating a relationship in Access . . . 98
Creating a relationship in SQL Server . . . 100
SQL Server views and Access queries . . . 102

Fundamental SQL . . . 103
Selecting all records from a table . . . 104
Selecting all records that meet one criterion . . . 104
Selecting all records that meet several criteria (using AND) . . . 105
Selecting records that meet one or more of several criteria (using OR) . . . 105
Useful SQL keywords . . . 106
COUNT . . . 107
SUM . . . 107
TOP . . . 108
BETWEEN . . . 108
ORDER BY . . . 109
IN . . . 110
GROUP BY . . . 111
DISTINCT . . . 112
Going on a DATE . . . 113
Making the connection . . . 113
Setting up a DSN to an Access database . . . 114
Setting up a DSN to a SQL Server database . . . 114
Connecting from Dreamweaver . . . 115
The Simple Recordset builder . . . 116
The Advanced Recordset builder . . . 119
Using commands . . . 122
Inserting a record . . . 124
Updating a record . . . 125
Updating multiple records (a simple example) . . . 127
Deleting a record . . . 128
Deleting multiple records (simple example) . . . 129
Conclusion . . . 131

Chapter 5 **Working with Forms** . . . **133**

Form attributes: Action and Method . . . 134
get method . . . 134
post method . . . 135
Retrieving form values with Request.Form . . . 135
Creating sample login forms . . . 135
Forms with text box elements . . . 136
Forms with list/menu elements . . . 137
Menu element selection form with conditional logic . . . 138
Forms with check box elements . . . 139
Receiving data from a URL parameter . . . 141
Retrieving URL parameters with Request.QueryString . . . 141
Sending form values to e-mail . . . 143
Setting up your SMTP server . . . 143
ASP mail components . . . 146
General requirements for mail components . . . 147
Sending e-mail with mail components . . . 148
Sending e-mail with CDO . . . 148
Sending mail with AspEmail . . . 151
Sending e-mail with JMail . . . 154

Dynamic e-mail interaction . . . 156
Sending e-mail by hyperlink . . . 156
Sending e-mail via a form button . . . 158
Making e-mail property values dynamic . . . 160
Sending e-mail with dynamic values . . . 160
Sending e-mail with dynamic form field values . . . 162
Sending a page to a friend . . . 164
Real-world examples using forms . . . 167
Sending a user a forgotten password . . . 167
Creating a mailing list . . . 170
Conclusion . . . 173

Chapter 6 **Building a Random Quote Generator** . . . **175**

Creating the quotes database table . . . 176
Building the form . . . 177
The Insert Record server behavior . . . 179
Adding conditional code . . . 180
The Repeat Region server behavior . . . 182
The Update Record server behavior . . . 186
The Delete Record server behavior . . . 189
Conclusion . . . 192

Chapter 7 **Completing the Quotes Administration System** . . . **195**

Updating the quotes database table . . . 196
Author administration . . . 199
Building the Insert Author page . . . 199
Category administration . . . 201
Building the Insert Category page . . . 201
Updating the Insert Quote page . . . 203
Creating the recordsets . . . 203
Building the Authors recordset . . . 203
Building the Categories recordset . . . 204
Building dynamic select lists . . . 204
Updating the Insert server behavior . . . 207
Updating the edit page . . . 208
Copying recordsets . . . 208
Adding authors and categories to the edit page . . . 210
Updating the Quotes recordset . . . 211
Binding dynamic select lists with a selected item . . . 211
Updating the Update server behavior . . . 213
The random quote generator . . . 214
Creating a join in the database . . . 214
Displaying a random quote . . . 216
Building the Quotes recordset . . . 217
Conclusion . . . 220

Chapter 8 **Restricting Access** **223**

Creating the administrators database table 224
Creating the login system 225
Building the login form 225
Adding the Log In User server behavior 227
Restricting access 229
Adding the Log Out User server behavior 231
Registering new users 233
Checking the new username 235
Creating a user-friendly login 236
Expanding the Log In User server behavior code 237
Implementing the "remember me" feature 239
Updating the login form 239
Baking the cookies 241
Conclusion 243

Chapter 9 **Creating a Blog** **245**

Blogging application overview 246
The database 246
The blog display 246
Administrative controls 247
Administrative login/logout 247
The search function 247
Creating your blog database 247
Creating the database tables 249
Table 1: tbl_onews 249
Table 2: tbl_users 250
Populating the database tables 250
Populating tbl_onews 251
Populating tbl_users 251
Setting user permissions for tables 251
Connecting Dreamweaver to your database 253
Connection strings 253
DSN 254
Displaying blog entries 255
Creating onews_main.asp 255
Designing onews_main.asp 256
Wiring up onews_main.asp 257
Creating onews_details.asp 261
Designing onews_details.asp 262
Wiring up onews_details.asp 262
Creating onews_archives.asp 264
Designing onews_archives.asp 264
Wiring up onews_archives.asp 265
Blog administration 266
Creating onews_admin_archives.asp 266
Designing onews_admin_archives.asp 267
Wiring up onews_admin_archives.asp 268

Creating onews_admin_details.asp . . . 270
Designing onews_admin_details.asp . . . 271
Wiring up onews_admin_details.asp . . . 273
Securing the blog administration pages and creating login/logout functionality . . . 276
Designing login.asp . . . 277
Wiring up login.asp . . . 278
Creating logout.asp . . . 279
Searching your blog . . . 281
Creating the search form . . . 281
Creating the results page . . . 281
Designing onews_searchresults.asp . . . 283
Wiring up onews_searchresults.asp . . . 283
Conclusion . . . 285

Chapter 10 **Image Gallery** . . . **287**

Overview of the image gallery application . . . 287
Storing information in the database . . . 288
Inserting, editing, and displaying albums . . . 288
Uploading and displaying photos . . . 288
Editing captions and deleting photos . . . 288
Creating the database for the image gallery . . . 289
Table 1: tbl_photoalbums . . . 289
Table 2: tbl_photos . . . 290
The relationship between tbl_photos and tbl_photoalbums . . . 292
Connecting Dreamweaver to your database . . . 294
Connection strings . . . 294
Data Source Name (DSN) . . . 295
Inserting, editing, and displaying albums in the image gallery . . . 295
Creating myphotos_main.asp . . . 295
Designing the main web page . . . 297
Implementing the dynamic features of the main web page . . . 298
Creating addalbum.asp . . . 301
Designing the Add Album page . . . 301
Wiring up the Add Album page . . . 302
Creating edit_photo_album.asp . . . 304
Implementing the dynamic features . . . 304
Creating pages for uploading and displaying photos . . . 307
Creating upload.asp . . . 307
Designing upload.asp . . . 308
Wiring up upload.asp . . . 309
Creating upload_action.asp . . . 312
Creating the Edit Caption and Delete Photo pages . . . 316
Creating edit_caption.asp . . . 317
Creating delete_action.asp . . . 320
Conclusion . . . 322

Index . . . **327**

ABOUT THE AUTHORS

Omar Elbaga has been passionately involved in web development for over six years, over which time he has completed several projects for small businesses and nonprofit organizations, including personal sites. He has previously coauthored two books on Dreamweaver web development, and maintains his own Dreamweaver tutorial/resource site, now known as `dmxfire.com`. Because he did not start out as a programmer himself, he has a knack for teaching dynamic web development to those who do not have programming backgrounds.

He is also a postsecondary language arts instructor, artist, independent filmmaker, and graduate student at New Jersey City University, currently finalizing his master thesis as a reading specialist. You can find some of his work in these areas and more information at `http://elbaga.net`.

Rob Turnbull is the senior developer for Lighthouse – design for business limited, an established new media design and marketing company based in Shrewsbury, UK. Clients across Europe, from small businesses to blue chip companies, provide an increasing workload, which includes the development of databases, websites, multimedia presentations, interactive CD-ROMs, promotional videos, and 3D artwork (animations and stills).

His personal website, `http://robgt.com`, is primarily focused on offering help and guidance to fellow Dreamweaver users, including tutorials and links to helpful resources and some useful extensions. You'll also find links to relevant books on web development and design, including his three previous books about Dreamweaver.

ABOUT THE TECHNICAL REVIEWER

Jason Nadon has been in the information technology field for the past nine years, and building web applications and solutions with Macromedia tools for the past six. He holds several industry certifications and is currently employed by Thomson Creative Solutions as a web services administrator.

Jason also manages the Ann Arbor Area Macromedia User Group and enjoys being an active member of the Macromedia developer community.

ACKNOWLEDGMENTS

Thanks to my mom and dad for raising me, supporting me, and pushing me toward fulfilling all of my educational goals and aspirations. Thanks to Uncle Ibrahim and Uncle Salah for being two of the greatest male role models in my life. Thanks to Aunt Khadiga and Aunt Hanem for being like second mothers to me. Thanks to my classy fiancée, Mona, for always being there, standing beside me and supporting me whether things are up or down, and always making my heart smile. Thanks to my brothers and sister, Tamer, Hesham, Soliman, and Yasmine—I couldn't ask for better siblings to have in my life! Thanks to my baby niece, Safiya; I can't wait to see you grow up. Also, thanks to all spouses and relatives of the people I have mentioned who have also been there for me. Thanks to the true and caring educators who taught me at NJCU, and my sincere and faithful teachers at McNair Academic High School. Thanks to all of my own students at Middlesex County College and Hudson County Community College. Thanks to Ihsan, and all my friends who stand by me; you know who you are. Thanks to you, dear reader, for entrusting me with your learning.

—Omar Elbaga

INTRODUCTION

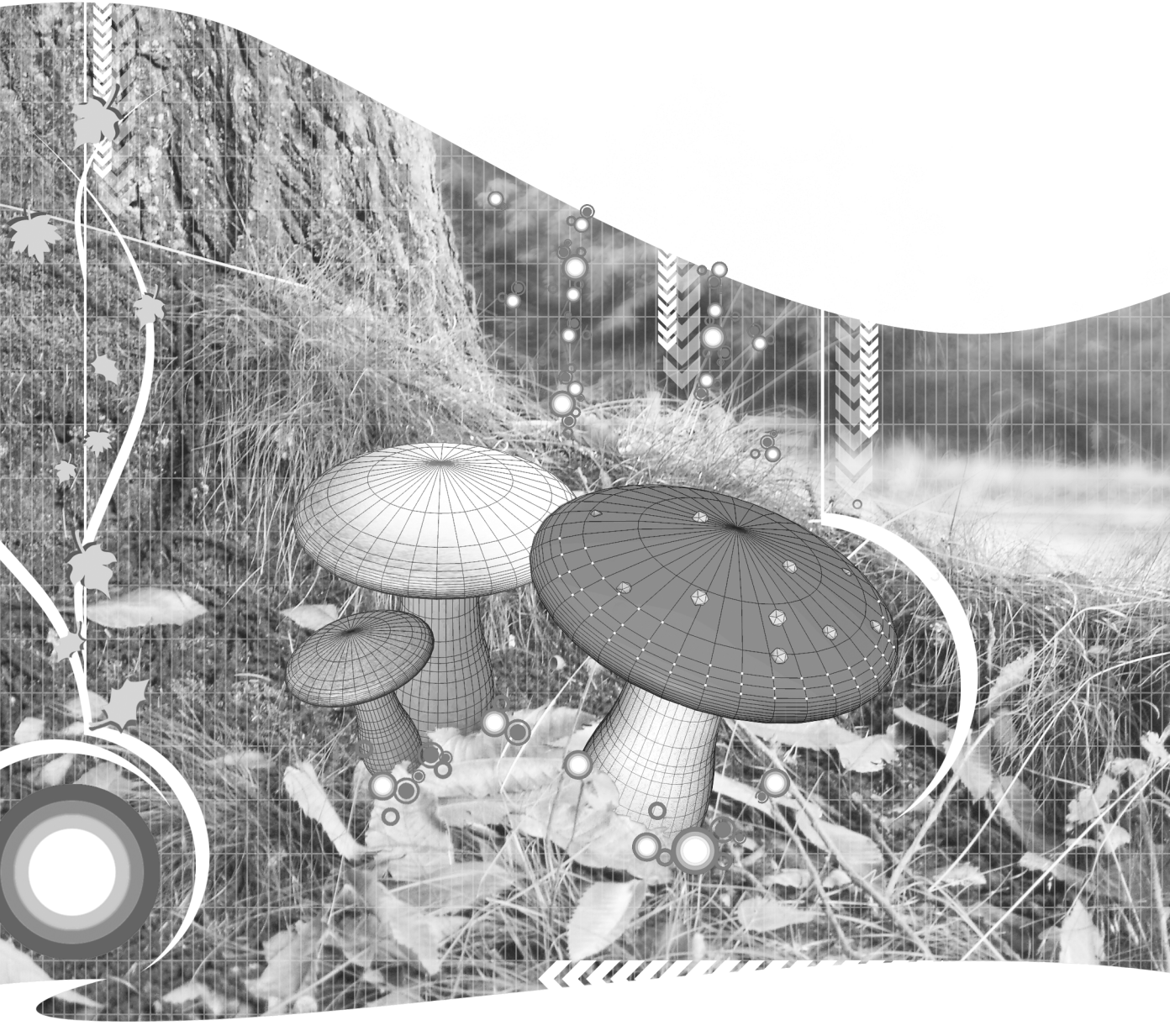
We're glad you picked up this book to begin learning about building dynamic websites using Dreamweaver 8 with ASP technology. This book is geared for both beginners who know little about dynamic and database-driven websites, and intermediate users who want to begin building more powerful web applications—such as blogs and photo albums—along with working administration systems to effectively manage those data-driven applications.

To complete all of the activities in this book, you will only need Dreamweaver 8, a server running Microsoft's classic ASP (Active Server Pages), and Windows 2000 or above. If you do not use the Windows platform, you can run the code remotely, such as on your web host's server—just make sure your web host allows ASP scripts to run on its server.

You should also have either Microsoft Access 97 or higher, or SQL Server 2000 or higher, to build the databases that you will connect your pages to. You can download a 180-day trial of the new SQL Server 2005 at `www.microsoft.com/sql`. Alternatively, you can make use of the new (and free) Express edition of SQL Server 2005, which is available from the same location. You can also download a 30-day trial of Dreamweaver 8 at `www.macromedia.com`.

It is possible that throughout your reading of this book, you will have some questions or comments, so feel free to drop the authors a note. Omar Elbaga can be contacted directly through the feedback/contact section of his website, `http://elbaga.net`, and Rob Turnbull can be contacted through the contact form on his website, `http://robgt.com`.

We have written this book for you, and we are here for you. Please give us time to respond due to the large amount of responses we have to deal with. We look forward to helping you learn more about building dynamic websites using Dreamweaver 8 with ASP!



Chapter 1

DREAMWEAVER AND DYNAMIC SITES

In this chapter, you'll learn about a number of important concepts, including what a dynamic website is and how it's different from a static site, what awesome dynamic features Dreamweaver offers you as a progressing web developer, and how ASP (Active Server Pages) fits into the mix. You'll also discover which ASP features are supported by Dreamweaver, where ASP features are located in Dreamweaver, and how Dreamweaver automates ASP code to create dynamic sites.

The first thing going through your mind is probably "What in the world is a dynamic website?" Well, I want you to figure this out with me. Take two minutes and jot down some things you know about the word "dynamic." What do you think a dynamic site is? And what do you think it can do? Do that, and then come back and continue reading. Remember, you're thinking of the word "dynamic" in the context of computers and web development.

All right, you're back! That was fast! What did you write down?

If you said something like "strong," you're right. If you said something like "powerful," you're also right; however, these meanings are a little different from what the word "dynamic" means in the context we're talking about. If you said anything like "changing" or "moving," you're right on target—that's exactly what it means!

You can also look it up in the dictionary—if you have access to the Internet, do a quick search at `www.dictionary.com` or `http://m-w.com` (Merriam-Webster Online).

According to the American Heritage Dictionary, dynamic means "characterized by continuous change, activity, or progress." That's it! It's all about change. See Figure 1-1 for further clarification on the definition.

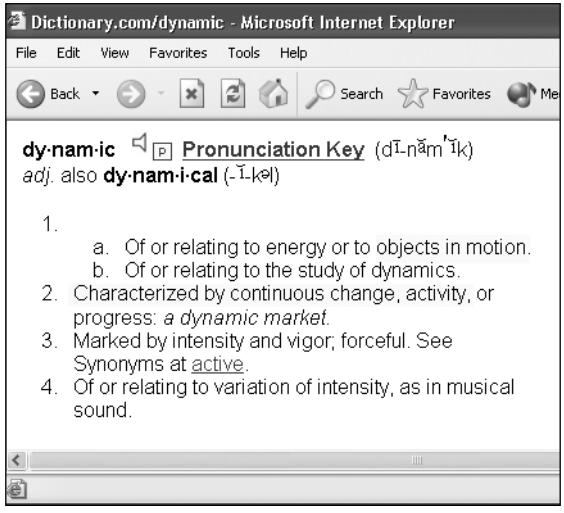

Dictionary.com/dynamic - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
dy·nam·ic Pronunciation Key (dī-năm'ĭk)
adj. also dy·nam·i·cal (-ĭ-kəl)
1.
a. Of or relating to energy or to objects in motion.
b. Of or relating to the study of dynamics.
2. Characterized by continuous change, activity, or progress: a dynamic market.
3. Marked by intensity and vigor; forceful. See Synonyms at active.
4. Of or relating to variation of intensity, as in musical sound.

Figure 1-1. Here is a screenshot of the definition of the word "Dynamic" at Dictionary.com. "Dynamic" means "changing."

So sit back, relax, and let's have some dynamic fun!

The first thing I want you to ask yourself is "What do I personally do with web development now?" and "What more do I want to be able to do?" If you said to yourself that you're writing back-end database models for Amazon.com, you might want to stop right here!

If you said something like one of the following, though, this book will definitely help you take your web development skills to the next level:

- I am designing websites for myself and family.
- I am working for a company whose managers want me to upgrade the site so that they can begin adding content to the site on their own.
- I want to start doing more dynamic things to my site, such as allowing web surfers to register and log in.
- I want to start collecting information from my site visitors.
- I want to start protecting site content from particular web surfers.

Data exchange

As a web developer, you need to make your website interesting and interactive. Thus, you need to tailor your website according to the web surfer. Shoving the same static data at all of your users is bound to leave someone (or many people) behind. The greatest problem with a static website is the

simple fact that it doesn't change. Imagine Amazon.com without the personal wish list, or without the capability of tracking your recent purchases, or without the feature that says "Customers interested in this title may also be interested in . . ." Dynamic websites can spontaneously reconstruct themselves according to an individual web surfer's needs. These sites change depending on how each user interacts with them. This is what data exchange is all about, and data exchange is what enables Dreamweaver 8 with ASP to save your website from static doom!

You want to exchange data between yourself and the web surfer; technically, the web surfer's computer will be doing the exchanging. Basic HTML can't handle this kind of data exchange, which is why we now use web programming technologies to do this for us—in our case, the particular technology is ASP. The web programming aspect takes care of all the dynamic stuff and then tells HTML what to print out. So instead of having to tell HTML what to do every time you want to change your data (by recreating HTML web pages), you set up an interactive system in which ASP tells HTML what to do. You're saying, "HTML, we've already designed you. From now on, let ASP deal with you for major changes in terms of data."

Let me show you a quick example. If you've already set up your environment, you should be up and running to view ASP pages on your local server. (You'll learn all about setting up the Dreamweaver environment in Chapter 2.)

1. Load Dreamweaver.
2. Choose File ➤ New, and select the General tab. Select Basic Page from the Category menu (on the left) and HTML from the Basic page menu (on the right), and click the Create button at the lower right of the window (see Figure 1-2).

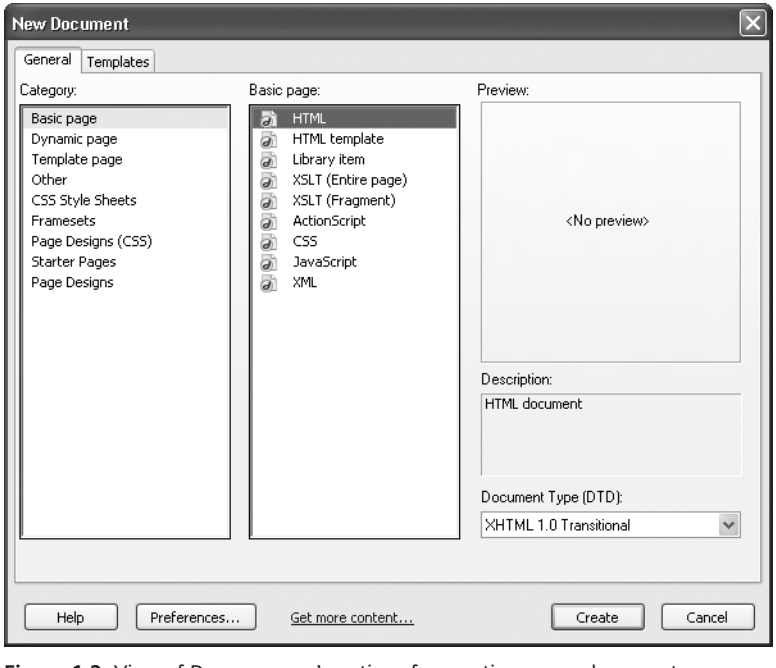


Figure 1-2. View of Dreamweaver's options for creating a new document

3. On your new "basic" page, type in the following message in design view:

   Hi, I don't know who you are because I'm just a plain ol' HTML web page.

   You might want to change the title of your document. I changed the title of my document to Plain 'ol HTML. View the page in your web browser by selecting File ➤ Save As, naming the file, and finally loading it into your browser. You can also press F12 to have Dreamweaver automatically load the page into your browser (see Figure 1-3).

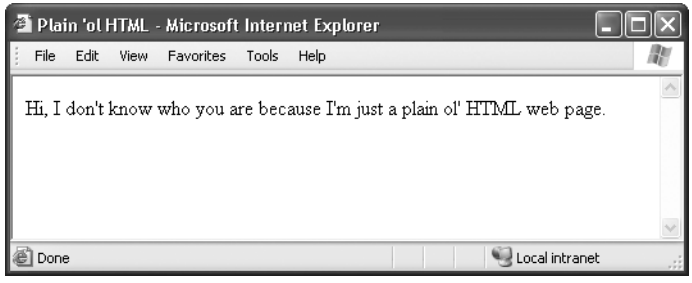

Plain 'ol HTML - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Hi, I don't know who you are because I'm just a plain ol' HTML web page.
Done
Local intranet

Figure 1-3. View of static HTML page in the web browser

4. Next, within Dreamweaver, select File ➤ New. Select the General tab, highlight Dynamic page from the left-hand Category list and ASP VBScript from the right-hand Dynamic page list, and click the Create button at the lower right of the window (see Figure 1-4).

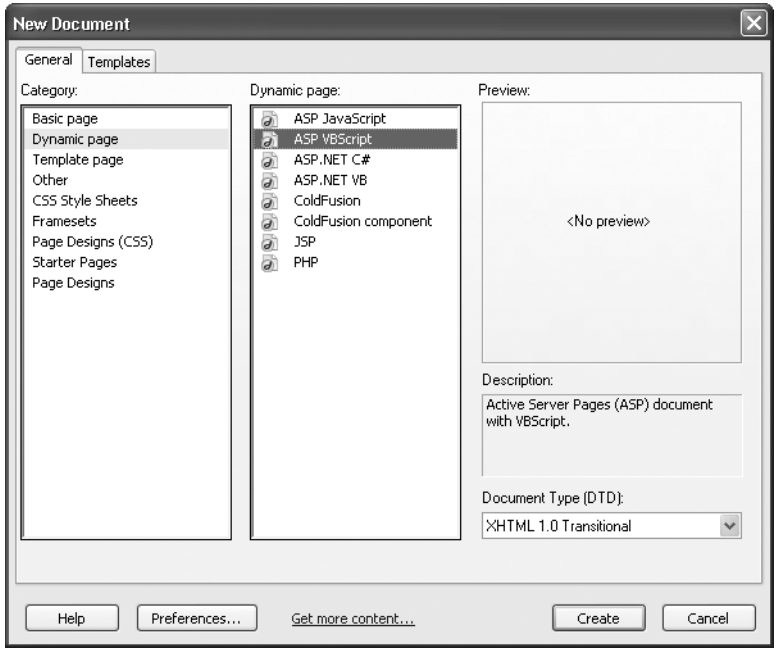


Figure 1-4. View of Dreamweaver's New Document options, in which Dynamic page and ASP VBScript are selected

Your cursor will now be within the design body of the document. From the standard toolbar, select Insert ➤ ASP Objects ➤ Output. Dreamweaver will now create the following ASP brackets for you: <%= %>. (We'll go into more detail about these brackets in Chapter 3.)

5. Within the brackets, type the following code:

```
Request.ServerVariables("REMOTE_ADDR")
```

In Code view, your final code should look like the following:

```
<%= Request.ServerVariables("REMOTE_ADDR") %>
```

In Design view, you'll see an ASP textual placeholder that represents the dynamic code. Highlighting the dynamic text in Design view will automatically highlight the same code from within Code view. Switch to Design view, take the focus off the dynamic text by moving your cursor to the right of the text, and hit Enter. Now type the following message:

```
I know who you are now. Your IP address is listed. Don't try anything funny!
```

6. Save your page as dynamic_asp.asp and view it in your web browser (see Figure 1-5).

Dynamic ASP - Microsoft Internet Explorer
File Edit View Favorites Tools Help
127.0.0.1
I know who you are now. Your IP address is listed. Don't try anything funny!
Done
Local intranet

Figure 1-5. View of the dynamic_asp.asp page in the web browser

See the difference? With HTML, you're unable to recognize the web surfer; you can only share *your* information with him or her. But with ASP, you can collect information from the user. In this vein, you can also have the web surfer provide you with specific information, which can be accomplished through forms. We'll go into further detail about this in the chapters to come, but most importantly, we want to drive the following point home:

> *ASP gives you the ability to interact with your web surfers by allowing you to exchange data with them. What's more, once you have that data from the user, you can update data on your web pages accordingly.*

With a static HTML website, your data is merely sent to the web surfer's web browser, unchanged and unprocessed, by a dynamic web programming language such as ASP (see Figure 1-6).

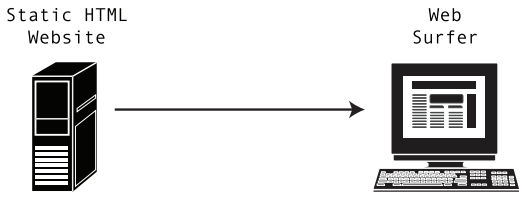


Figure 1-6. Illustration of data flow from a static site

With a dynamic ASP website, data can be sent back and forth between the website and the web surfer's web browser (see Figure 1-7). As a web developer, you can do a variety of things with the data, such as collecting it for statistical purposes or processing it with a dynamic web programming language such as ASP. In the latter case, ASP processes the data on the server and then recreates the HTML page, sending the data back to the web surfer's browser with new data.

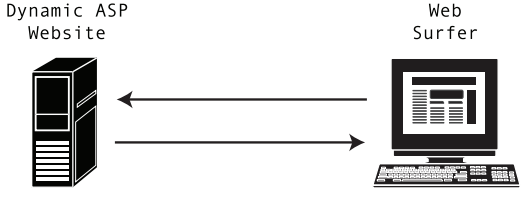


Figure 1-7. Illustration of how data is exchanged between a dynamic site and the web surfer

Data that you provide, and data you collect from the web surfer's computer, can be stored in a database. The ASP page can then connect and interact with the database while interacting with the web user. The flow of the interaction is all up to you (see Figure 1-8). For example, you can send data that you specify (such as contact info or "about" info) from a database to the web surfer's browser without allowing the user to alter the data, or you can actually allow the surfer direct access to the database so they can add, edit, and delete data.

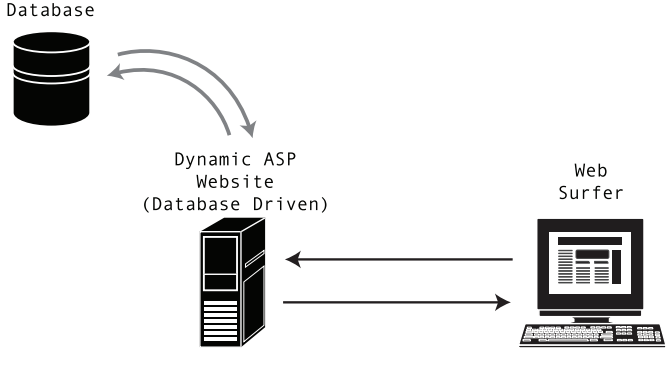


Figure 1-8. Illustration of database-driven data exchange between a dynamic site and the web surfer

Dynamic takes over static

What do I mean by dynamic "taking over" static? I mean that we're now in a new era of web development and we need to pick up the pace and get with the times. Web development has been zooming right by us, and you're just sitting around doing nothing! Well, not you, since you picked up this book.

Web development has progressed from displaying simple pictures to streaming live video footage. Even more drastically, it has gone from displaying basic text to regenerating huge reports on the fly, in seconds, at the touch of a web button. It has gone from static text to dynamic text—for example, you were once able to write up your own resume, create a website with basic HTML pages, and upload it to the Web. Now, websites are able to generate your resume and a thousand others on the fly, at the same time, based on the simple submission of a couple of online forms! If you don't believe me, go visit Yahoo's Hotjobs (`http://hotjobs.yahoo.com`) or `www.monster.com`.

Dreamweaver server models

How exactly does Dreamweaver make your sites dynamic? In the past, you could create HTML pages through Dreamweaver's powerful visual layout system, but you would then have to edit the content of the actual HTML page and hand-code the HTML tags over again to make the pages more dynamic. Now, Dreamweaver integrates what Macromedia calls server models. You, as a web developer creating websites with Dreamweaver, get to choose whichever server model you want to create dynamic websites with, almost seamlessly. And the best part about it is that Dreamweaver generates the web programming code for you through its robust internal storage of stock code. It does so based on the server model you select.

The ASP VBScript server model

When developing web applications in Dreamweaver, it's a good idea to choose a server model that you're familiar with, or at least one that you want to become more familiar with. When you use server models, Dreamweaver will generate full-blown programming code—but you'll want to be able to know what Dreamweaver is doing, and you'll also want the ability to tweak code to attain more desirable results. As such, this book will familiarize you with ASP as a language before teaching you how to develop using Dreamweaver's drag-and-drop ASP server behaviors.

We'll show you how to set up your environment in the next chapter—but for now, fire up Dreamweaver. Let's take a look at the server models that Dreamweaver supports.

1. From the toolbar at the top, select Site ➤ New Site.
2. Select the Advanced tab, and then highlight Testing Server in the Category list.

3. You'll see a drop-down list of the server models. Take a look at all of the server models Dreamweaver supports (also shown in Figure 1-9).

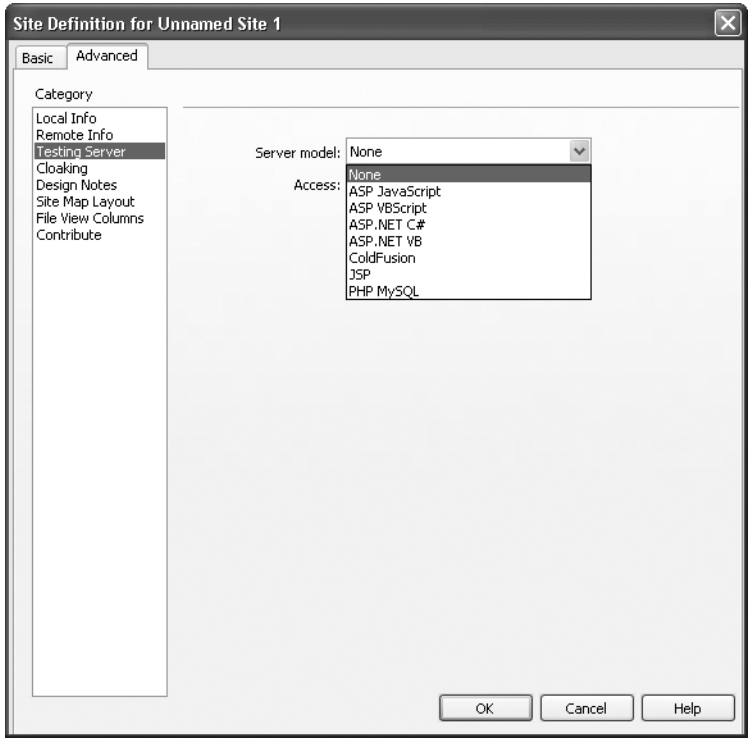


Figure 1-9. The listing of server models Dreamweaver supports

ASP can be coded using either JavaScript or VBScript. Throughout this book, you'll be using ASP with VBScript, not JavaScript. The reason for this is that VBScript is much easier to code with. JavaScript is much more sensitive—it's case-sensitive, which can drive a programmer crazy, let alone people like us. One incorrect case and the entire web application can go bananas. Even more importantly, there is loads of free information on ASP with VBScript; but it's hard to get support with ASP coded with JavaScript because it's rarely used. VBScript is the most popular language to code ASP pages in, and you'll find tons of resources on the Web. Go ahead and do a Google search if you don't believe me, but I know you trust your teacher!

> *Throughout the book, you'll be working with the ASP VBScript server model. Since Dreamweaver automatically generates all the code for you, it's technically possible to use the ASP JavaScript built-in server behaviors (or any server model for that matter) for these exercises. As you progress through the book, however, you'll be enhancing Dreamweaver's automatically generated code, as well as the VBScript code itself, so it's important that you stick with the ASP VBScript server model throughout.*

What Dreamweaver 8 offers you as an ASP developer

Earlier, I said that ASP makes web pages dynamic by allowing the exchange of data between your website and the web server. ASP does this in two main ways. One way is through ASP's own unique language, which makes use of conditional logic, functions, sessions, and cookies. (You'll learn more about all of these in Chapter 3.) Another way ASP does this is through connecting to databases.

Checking out data

Regarding the first way mentioned, ASP can respond to a user's interaction in various ways depending on the criteria you specify. For example, you could have ASP detect which browser your web surfer is using to view your site, and then you could have ASP automatically redirect the user to a different page of your choice. Before ASP, you couldn't do this sort of thing with ease—now, you no longer have to be there when it's all happening. You can set the conditions and then let ASP do the work. (You'll learn more about this in Chapter 3.)

With ASP functions, you can verify data submitted through a form on your site and have ASP respond accordingly. For example, say you want to make sure that a user reads a "Terms and Conditions" page before proceeding on your site. In such a case, you can make sure the user ticks an appropriate check box before submitting a form—if the user fails to do this, you can redirect him or her to an error page. More generally, you can make sure the user provides you the data you need in order for him or her to go on interacting with your site.

Connecting to databases

The second way that ASP allows the exchange of data is one of the most unique and fun features about Dreamweaver: ASP can actually connect to a database, and Dreamweaver automates the whole process for you. Being that ASP code itself can qualify data, imagine how much more ASP can do when it has the ability to connect to a database. Now you're talking about qualification on a much grander scale.

As you know, databases can store huge amounts of data efficiently. Databases make it easy to store, edit, and view data once an efficient database schema is set up. I'm sure you're familiar with Microsoft Access—this is one of the most popular database systems used around the world, but it's mainly used for local office use, and not to store huge amounts of data, because it's simply not as robust as others. When it comes to companies that need to store massive amounts of data, SQL Server is more popular.

Anything you can do with databases on your computer locally you can now do online with ASP—not only querying data, but also inserting, editing, and deleting it. Now that's dynamic. For example, instead of having to create a static HTML web page for every one of your products, you can create one master template—that is, one ASP page that connects to the database and has the ability to display any product selected by the user. The ASP code requests the desired item from the database and simply inserts it into the ASP page, preventing you from having to create a separate HTML page for each product.

Where is ASP in Dreamweaver?

Great question! Now you'll take a quick look at where the heck your ASP stock code is located within Dreamweaver, and how you access it.

ASP features supported by Dreamweaver

Anything you can do in ASP can be done within Dreamweaver. Dreamweaver's versatile page layout options allow you to view your pages in three different views: Design, Code, and Split (shown in Figures 1-10, 1-11, and 1-12, respectively). Design view allows you to use Dreamweaver's built-in functions for HTML, JavaScript, and ASP. Code view gives you the option to insert ASP code directly onto the page. Split view allows you to switch easily between Design and Code view, or have a look at both views at the same time. This is helpful when you want to make changes to code directly and see instantly how they affect the design of the page.

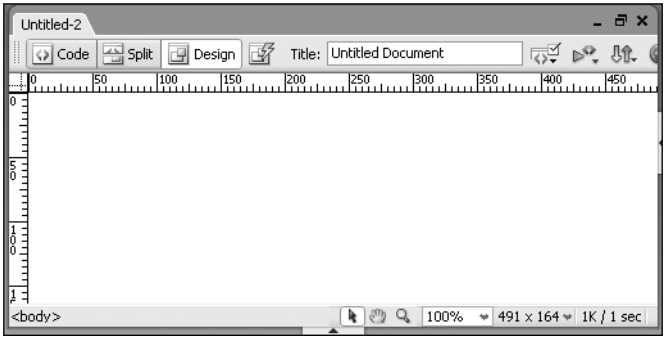


Figure 1-10. Example of Dreamweaver page development in Design view

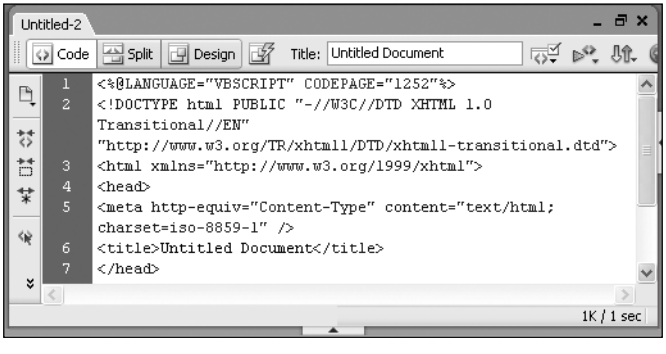


Figure 1-11. Example of Dreamweaver page development in Code view

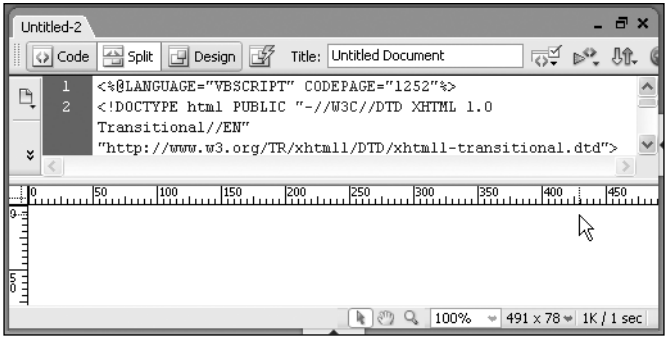


Figure 1-12. Example of Dreamweaver page development in Split view

Before you dive into any ASP, which will naturally be a bit foreign to you, you'll briefly look at two main sections of Dreamweaver: the Insert bar and the Application panel.

Insert bar

Once you select the server model you'll be using—ASP VBScript—you can gain access to ASP from two main sections in Dreamweaver: the Insert bar and the Application panel.

Let's take a look at the Insert bar first, which is located above the Dreamweaver document window and below the standard toolbar. It's displayed by default—but to be sure it's open in your Dreamweaver window, go to View ➤ Toolbars and make sure Insert is checked. Once it's open, you can view the Insert bar in two different styles. The first style is the Show as Menu view, in which each category in the bar is displayed separately (this is Dreamweaver's default style). With this view, you have to click your selection from a drop-down menu on the left-hand side of the Insert bar each time you want to view a new category. Figure 1-13 shows this style, in which the Application category is displayed.

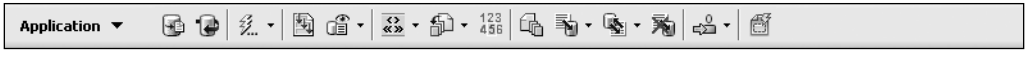


Figure 1-13. The Application category of the Insert bar in the Show as Menu view

If you click the Application text, you'll see a menu displaying the other elements.

The other Insert bar style, which I personally prefer, is the Show as Tabs style. In this style, elements are grouped as tabs that form one panel (see Figure 1-14). You can select this style by clicking Application on the Insert bar, and then clicking Show as Tabs. This view allows you to switch between different elements easily.

Figure 1-14. The Insert bar in the Show as Tabs view

Switch to the Show as Tabs view, and try clicking the ASP tab Your Insert bar should now look like Figure 1-15.

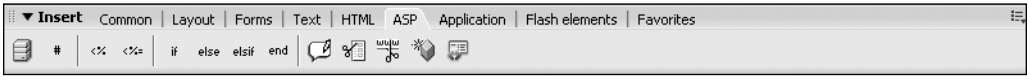


Figure 1-15. The ASP tab of the Insert bar in the Show as Tabs view

You can also access these same menus through the standard toolbar by selecting Insert ➤ Application Objects and Insert ➤ ASP Objects. This is where the majority of your ASP functionality is located.

Before I describe the other place Dreamweaver stores ASP functionality—the Application panel—let's take a brief look at what's stored in the ASP and Application tabs of the Insert bar.

The Insert bar's ASP tab

The Insert bar's ASP tab stores some of your basic ASP hand-coding functionality. Using this menu will help speed up some of your basic hand-coding. This menu doesn't store your major ASP dynamic functionality, however. You won't use it that often because it only automates some basic text for you. You should still try it out, though. By the time you get through Chapter 3, you'll be able to write all of this code yourself with ease. This menu just helps speed up the process slightly.

In the sections that follow, I'll briefly describe the functions of each icon on the ASP tab, from left to right. (You can see the name of each button in a tooltip if you hover over the icon.) Note that some of the buttons are described under one heading due to their relationship.

Server Variables. Server variables hold information about your server—the computer server your web pages are hosted on. This information includes data such as IP address, type of computer, location of your web pages, names of files and folders, and so on. ASP can access these variables using code such as the following, which prints out the server's IP address to the screen:

```
<%= Request.ServerVariables("REMOTE_HOST") %>
```

Dreamweaver stores the different code needed for your convenience so you don't have to remember it.

Include. Includes, also known as Server Side Includes (SSI), are simply pieces of code saved as separate files and embedded into other web pages. They come in handy very often—for example, say you have some code that you need on every page, such as a footer including a company name and a date. The date can be automatically updated using an ASP function—but what if you want to update the company name? If the company name is stored in an include file, you just need to change the instance in the include, and the name will be updated on every page.

Dreamweaver uses ASP includes with the code it generates to connect web pages to databases because every page in a dynamic site generally needs to connect to the same database(s). Since the code for the connection doesn't change, Dreamweaver creates one document with the code for the connection and then just adds ASP includes on the pages you specify for the connection; then those pages will reference the code from the original document.

Code Block. This is Dreamweaver's way of adding a new, blank block or line of dynamic code to your web page. All ASP code must be within ASP delimiters (<% and %>). Here's an example of how these delimiters are used in code:

```
<% ' ASP code goes here %>
```

These delimiters are similar to HTML tags. If you're familiar with HTML, you know that all HTML content must appear within brackets, as in the following code:

```
<p>This is my web page!</p>
```

Dreamweaver will add the ASP delimiters for you when you want to hand-code a block of ASP code. You can either type out the ASP delimiters or you can just click the icon labeled Code Block, and Dreamweaver will write the delimiters out for you. Then you simply write your ASP code within the delimiters.

Output. The Output button inserts the <%= and %> delimiters, as shown in the following code:

```
<%= "ASP forever" %>
```

The Output button functions similarly to the Code Block button, but it also includes an equals sign after the first delimiter. The addition of the equals sign tells the HTML to print to the screen whatever is within the code block. The opening bracket along with an equals sign is actually an ASP shortcut for another, longer object, `Response.Write`, which tells the HTML to print to the screen.

Response.Write. The equivalent of the Output button code using the full `Response.Write` syntax is as follows:

```
<% Response.Write "ASP Forever" %>
```

If, Else If, Else, End. This is actually part of VBScript's conditional logic. It's used to tell the ASP processor to execute code based on conditions. When you use this code, you tell the ASP processor something like the following: `If` this is true `Then` do this, `Else` do this, `End`. You'll see exactly how to do this with real coding in Chapter 3.

Server.CreateObject. You don't need to worry much about this right now, but this is code that's needed to create an instance of an ASP object. You'll be using one of these objects for sending emails through your ASP pages, because the code for triggering emails will require that you create an instance of the `mail` object. You'll learn more about this in Chapter 5.

Trimmed Form Element. This code will get rid of all empty spaces before and after any text that's submitted through a form element, upon retrieving the data. This is important because you don't want to have white space taking up unnecessary room if this code block is to be submitted into your database.

Trimmed QueryString Element. Just as with the Trimmed Form Element code, this code trims off empty spaces, but it does so to the value of querystrings. Querystrings are made up of the data attached to a URL after a question mark (?). For example, in the following code, `user_id=56&username=Bob.` is the querystring.

```
http://www.elbaga.net/welcome_user.asp?user_id=56&username=Bob.
```

The code ensures that no empty spaces are received for the value of the `user_id` and `username` variables. Again, you'll learn more about querystrings in Chapter 3.

The Insert bar's Application tab

Now switch to the Application tab from the Insert bar so you can take a look at the ASP functionality there. The Application tab is where you'll be using the majority of the ASP functionality, and where most of the dynamic functionality is located. Figure 1-16 shows the Application tab.

Figure 1-16. The Application tab from the Insert bar, which contains 14 elements

Next, I'll briefly describe each element on the Application tab, from left to right.

Recordset. A recordset is a set of records stored in a database. Since ASP can connect to a database, you can retrieve records from any database table(s) you specify in any order you specify. This is the foundation of your database-driven ASP web pages. Most other dynamic ASP functions within Dreamweaver will not work without having at least one recordset specified in your document, because you first need to retrieve data before you can display or edit it. However, you won't necessarily need to create a recordset when you simply want to insert data into the database.

> *A recordset retrieves records from your database and makes them ready for you to display and manipulate in your web pages.*

Command. Simply put, the `Command` object will contain SQL instruction to be executed against the database (e.g., inserting, updating, or deleting a record). You'll learn more about SQL commands in upcoming chapters.

> *SQL (Structured Query Language) is a language that allows you to access and make modifications to a database.*

Dynamic Data. Dynamic data is any data that comes from a database or that's passed through ASP variables. You can insert this data anywhere in your document as regular text, you can have Dreamweaver create an HTML table to display it, or you can have Dreamweaver attach it to a form element.

Repeated Region. A repeated region is used for cycling through a number of records retrieved from a database in a recordset and displaying them on a web page. For example, say you've retrieved a list of products in a recordset. ASP will only display one product, unless you use a repeated region, which allows you to loop through all the products and display each one in turn. A repeated region also allows you to specify how many records from a recordset you want to show at once; if you have hundreds of products, it makes sense to browse them a few at a time.

Show Region. The Show Region button allows you to show or hide regions of data based on criteria you specify for a recordset. For example, suppose a user is searching through your database for a record. If the record isn't found, the recordset will come up empty. You can then show a particular message to your user when the record isn't found, such as, Sorry, your query returned no results. It can also be used to make extra functions available to someone logged in as an administrator as opposed to a regular user. This feature is versatile and can help make your web applications much more user-friendly and intuitive.

Recordset Paging. This feature gives you the ability to page through records. Dreamweaver will add the ASP functionality to allow the user to go to the previous, next, first, or last record. This works only when you're not displaying every record in a database table. For example, say you retrieve 100 records in your recordset—how will you display all of those records on a page? Will you display them all at once in a repeated region, or will you group them? This question goes back to style, but also usability. It's common to group that many records in tens or twenties so that a user can view 10 or 20 records at a time. The Repeated Region behavior allows you to show a specified amount of records from a recordset. After using this behavior, you can apply the Recordset Paging behavior to allow your user to page through the recordset.

Go To. This behavior makes it easy to pass values from your database to a page of your choice. With this behavior, you're given the option to pass the values through form elements or through the querystring of a hyperlink. This is great because it's like a filtering system. You can then make a detail page that would query the database to retrieve a matching value from within the database.

Recordset navigation status. This behavior displays a numerical count of how many records were retrieved in a recordset. This is helpful in various ways, because it provides the user with an idea of how many records exist in a database query. For example, if your user searched for something in your database, it's helpful to tell him or her how many records were found. Alternatively, you as the web developer might want to know how many users are in your database. In this case, you can just add this behavior to a document, and each time it's accessed, it will display the current number of records in a recordset without having to display every record on the page.

Master Detail Page Set. This behavior is similar to the Go To behavior, but instead it creates a set of two pages: one master page (the Go To page), and one detail page, which fetches whatever values were requested through the master page.

Insert Recordset, Update Record, Delete Record. These three behaviors do exactly what their names say. They will create the ASP code needed to insert, update, or delete records in a database. Keep in mind, once you know the SQL commands for inserting, updating, and deleting, you can use the Command behavior to do the same thing. The difference between using the Command behavior and using these behaviors is that these behaviors insert, update, or delete records through stock ASP code. Using your own SQL commands with the Command behavior does the same thing a little more efficiently, though, because it doesn't have to produce so much code.

Also, you need to know that you must actually retrieve a record before you can update or delete it (this isn't the case for inserting a record). This usually requires the use of the Go To or Master Detail Page Set behavior. Inserting a record, on the other hand, will simply insert the record to the database table of your choice.

User Authentication. This consists of several behaviors that allow you to authenticate your users. Dreamweaver will automate the following types of web applications for you: logging in a user, logging out a user, restricting access to a page based on your own criteria, and checking to see if a username already exists in a database table before allowing a web surfer to submit his or her username choice so that the username isn't duplicated for two different people.

XSL Transformation. This option allows you to integrate XSL transformations to any XML data residing on the server. XSL transformations will not be discussed in this book.

Application panel

The Application panel is the second major section that stores a great amount of dynamic ASP functionality. It's divided into three tabs: Databases, Bindings, and Server Behaviors. If you see another tab labeled Components, don't worry about it, because it isn't active for the ASP VBScript server model. You could also remove the tab to free up some space, but remember to put it back if you're using a different server model. You can do so by selecting Window from the standard toolbar and checking Components. The Application panel is an area that gathers related functionality in one spot. The server behaviors listed in the Application panel give you access to the same behaviors in the Application tab of the Insert bar (which were described in the previous section). Figure 1-17 shows an example of the Application panel.

Figure 1-17. The Application panel, which is comprised of four tabs

Next, I'll briefly describe the Databases and Bindings tabs on the Application panel.

The Databases tab

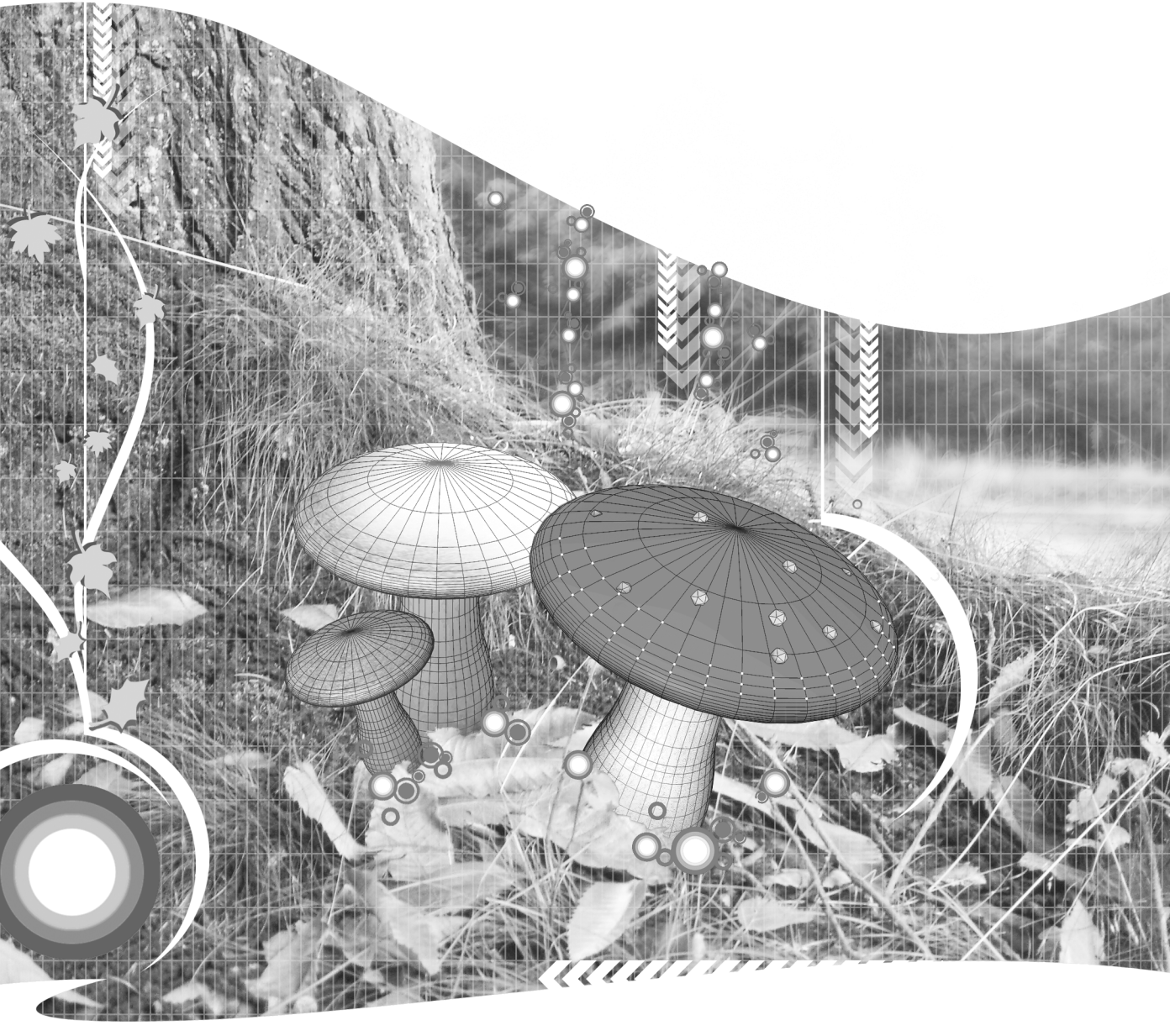
This is where you connect your ASP pages to your database. Dreamweaver creates all the necessary code for you once you enter the connection string. The code is then automatically inserted on each new dynamic page created. All the database connections related to your site are accessible through this tab. This is discussed in more detail in Chapter 6.

The Bindings tab

The Bindings tab consists of a list of any dynamic data you add to your document, such as recordsets, commands, and ASP variables. It's called Bindings because it allows you to bind dynamic data to selected areas of your document, such as form elements, static text, and HTML attributes. You can bind dynamic data that comes from a database or an ASP variable. For example, when having users update account-related data, you can automatically bind data already stored in the database to the text field of an update form so users don't have to retype all the data every time they want to update a single item related to their account (e.g., their password or the city and state they live in). This will be discussed further in Chapter 3.

Conclusion

You've only begun your journey into becoming a dynamic, database-driven, Dreamweaver web-developing guru! There's much more to come in the following chapters. The purpose of this chapter is to give you a general overview of the endless and unique possibilities for web development with Dreamweaver using the ASP VBScript server model. In upcoming chapters, you'll learn how to actually implement the knowledge you've attained from this chapter and create real-life web applications.



Chapter 2

THE DREAMWEAVER ENVIRONMENT

In this chapter, we'll discuss the features of the Dreamweaver environment that you should be aware of before you start to create your websites. We'll begin by examining the workspace layout, and then we'll move on to cover the panels and features of the Multiple Document Interface and the various toolbars available to make your work easier. Then we'll look at the new tools available in Design view before we wrap up the chapter with a demonstration of how to define an ASP VBScript website.

Workspace layout

When you first run Dreamweaver, you are asked to choose a layout, either Designer or Coder. There are only a couple of fundamental differences between the two:

- **Designer** layout places the side panels to the right of the main document window and opens documents in Design view by default.
- **Coder** layout places the side panels to the left of the main document window and opens documents in Code view by default.

In both layouts, you are able to drag and drop panel groups to the left and right of your workspace or undock them completely in order to get exactly the working environment you want. The way you arrange your panel groups is entirely up to you.

Figure 2-1. The new Workspace Layout menu items

A new, much beloved feature in Dreamweaver 8 is that you can now save your own workspace layouts. Once inside the program, you can change the layout by selecting Window ➤ Workspace Layout (Figure 2-1).

You'll notice a third standard option on the list of workspace layouts, Dual Screen, which is for those with dual monitor setups. The other two options allow you to save a workspace that you have set up under a name you specify and also to manage the layouts that you create (i.e., rename or delete them from your list of saved layouts).

The Multiple Document Interface

The **Multiple Document Interface**, or **MDI**, has remained structurally the same for this latest release of Dreamweaver, but a few items have been merged or moved around to make the workflow a bit smoother or more logical.

The main change has been in the creation of the unified CSS panel, which is now an all-encompassing panel showing the cascade of styles applied to a specific element, making it easy to see which attributes are being styled and how the cascade of the CSS applies to the tag selected on the page. You can also edit the styles directly in the panel and see your changes applied instantly to the page.

Another sensible change involves the Reference panel being moved to the wider Results panel at the bottom of the screen, making it much easier to read the panel's content (Figure 2-2).

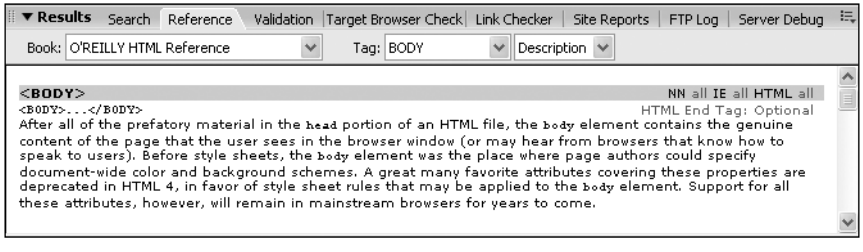

Results
Search
Reference
Validation
Target Browser Check
Link Checker
Site Reports
FTP Log
Server Debug
Book: O'REILLY HTML Reference
Tag: BODY
Description
<BODY>
NN all IE all HTML all
<BODY>...</BODY>
HTML End Tag: Optional
After all of the prefatory material in the head portion of an HTML file, the body element contains the genuine content of the page that the user sees in the browser window (or may hear from browsers that know how to speak to users). Before style sheets, the body element was the place where page authors could specify document-wide color and background schemes. A great many favorite attributes covering these properties are deprecated in HTML 4, in favor of style sheet rules that may be applied to the body element. Support for all these attributes, however, will remain in mainstream browsers for years to come.

Figure 2-2. The Reference panel contains several very useful reference books.

You can still group the panels in whichever way you prefer and, of course, you can rename most of the panel group names by right-clicking the panel group and selecting Rename Panel Group (Figure 2-3).

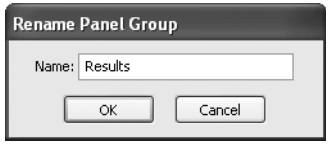


Figure 2-3. The Rename Panel Group dialog

Each panel has a context menu that can be accessed by right-clicking the panel tab or left-clicking the Panel Group Options menu icon (Figure 2-4).

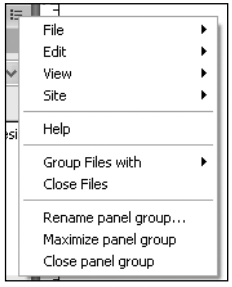


Figure 2-4. Click the Panel Group Options menu icon to access this context menu, where you are able to change panel grouping preferences.

There are a couple of restrictions that you should bear in mind when it comes to panel groupings: You cannot regroup the Properties panel and the Insert panel, nor can you group anything with them. However, you can group all other panels as you like.

To change the panel group in which a panel appears, use the Group [panel] with context menu option. This is also the method you would use to arrange the panels within the same panel group. When you group a panel within a panel group, it is added to the right-hand side of the list of tabs for that panel group. To shuffle the panel left, you regroup the other panels within the same panel group to add them to the right. This process can be bit of a pain, but if you must have the panels in a specific order, it's the only way to do it.

Nineteen panels make up the Dreamweaver environment. Fortunately, you don't have to have them all open at once. And now, because you're now able to save your own layouts in Dreamweaver, there's no need to have loads of panels open all the time, which results in more screen real estate for those panels that you use most and do need to have open.

In the sections that follow, we'll delve into each of the Dreamweaver panels you'll use most often.

The CSS panel

This new release of Dreamweaver is heavily geared toward CSS, which is apparent with the new unified CSS panel (Figure 2-5). It's a lengthy panel that includes everything about the CSS styles you're using when editing your pages.

A toggle button at the top of the CSS panel allows you to switch between looking at All styles available on the page and styles that apply to the Current selection on the page.

In the All view, the panel lists all the styles in the stylesheet under a heading of All Rules, and below that list it displays the properties for the style selected in a list labeled Properties.

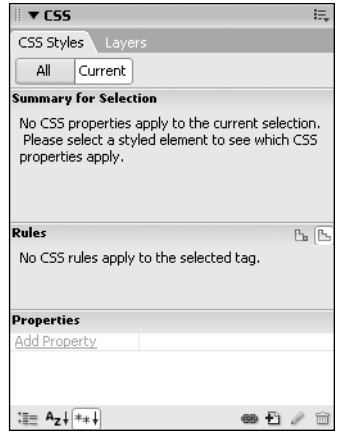


Figure 2-5. The unified CSS Styles panel

In the Current view, the panel is split into three sections. The first section, Summary for Selection, displays a summary of styles applied to the currently selected tag on the page. Below that is the Rules section, which displays a list of CSS rules that apply to the selection. And finally, below the Rules section is the Properties section, which displays the properties for the selected rule. You can also edit the properties for the selected rule in this section.

The Application panel group

Below the CSS panel is the Application panel group, which houses four panels: Databases, Bindings, Server Behaviors, and Components (Figure 2-6).

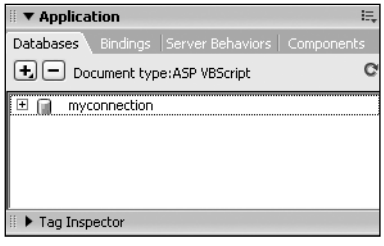


Figure 2-6. The Application panel group

Apart from the Components panel, which is not used for ASP sites, we'll be using these panels in this book quite a lot, so let's take a good look at them now to get familiar with them.

The Databases panel

The Databases panel displays your database connection information. If you have a connection to a database configured in Dreamweaver, this panel will display it. If you expand the connection, Dreamweaver goes to the database and brings back all the objects that it finds in there, grouping them under three items: Stored procedures, Tables, and Views (Figure 2-7). This panel assists you in creating database-driven, dynamic websites.

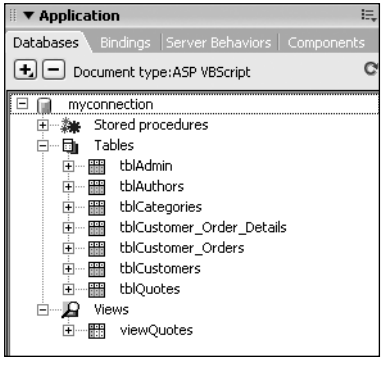


Figure 2-7. The Databases panel of the Application panel group shows the database connection(s) used in your site. Expand the connection to show the database objects available through it.

You can also define a new connection to a database from this panel by clicking the + button and selecting how the connection should be made, either through a custom connection string or by using a Data Source Name (DSN), as shown in Figure 2-8.

> *For further information about connecting to a database from Dreamweaver, please see the section "Making the connection" in Chapter 4.*

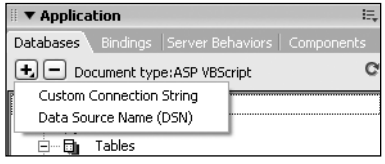


Figure 2-8. Creating a connection to a database

The Bindings panel

The Bindings panel allows you to create data sources that you can then bind to your page, including recordsets, commands, request variables, session variables, and application variables (Figure 2-9).

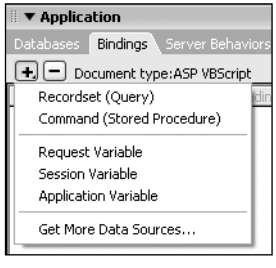


Figure 2-9. The Bindings panel and its menu of options

Once you have created a recordset, or any of the other data source items, it will be listed in this panel for you to bind to your page and optionally add formatting to (Figure 2-10).

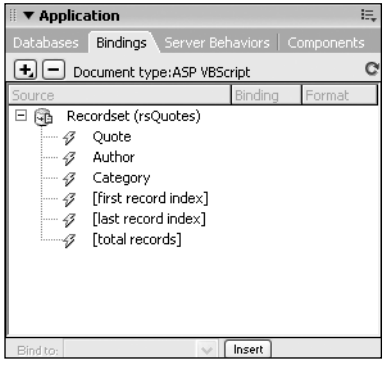


Figure 2-10. A recordset displayed in the Bindings panel

When you bind an item from the Bindings panel to the page, selecting that item on the page will expand the item in the Bindings panel to show the options available to you for that binding. Clicking the arrow icon in the Format column displays a large array of formatting options to suit lots of different purposes (Figure 2-11).

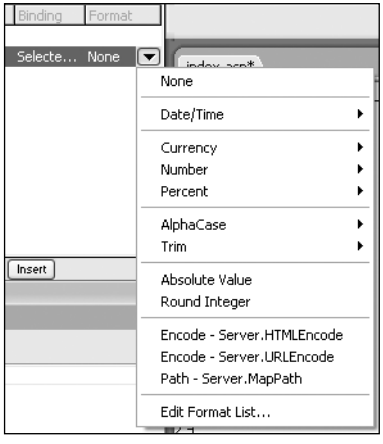


Figure 2-11. Many formatting options are available through the Bindings panel.

The Server Behaviors panel

The Server Behaviors panel displays all the server-side logic that you have applied to the page (Figure 2-12), including recordsets, commands, and any server behaviors, of course!

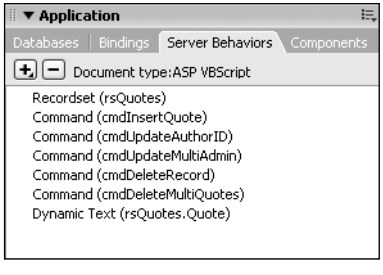


Figure 2-12. Displaying a list of items in the Server Behaviors panel

Double-clicking an item in this list will launch the appropriate dialog for you to edit that item's details. For example, if you double-click a recordset, the Recordset dialog will launch.

Many options are available from the + menu in the Server Behaviors panel, including entries that allow you to create a recordset or add a command to the page, duplicated from the Bindings panel (Figure 2-13).

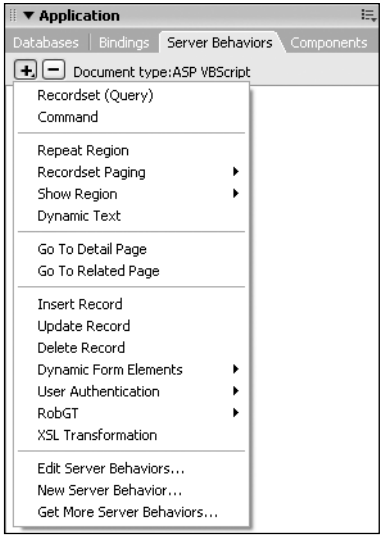


Figure 2-13. The Server Behaviors panel menu options

The Components panel isn't used in ASP VBScript sites, so you can close that panel to make room for something else you might use more readily, or to simply have less clutter in your environment.

The Tag Inspector panel group

The Tag Inspector panel group holds two panels by default: Attributes and Behaviors. The title of this group changes depending on what is selected in the document you're editing. If nothing is selected, then the name displayed is Tag <body>, meaning that you're at the top level of your document: the <body> tag (Figure 2-14).

Figure 2-14. The Tag Inspector panel group as displayed with nothing selected on the page

If you select something on your page, the title will change to reflect that selection. For example, selecting a paragraph of text on your page will make the title of this group Tag <p> (Figure 2-15).

Figure 2-15. The Tag Inspector panel group as displayed with a paragraph item selected on the page

This is to let you know, at a glance, what tag you will be affecting with any changes you make in this panel group.

The Attributes panel

The Attributes panel displays the properties of the tag you are editing in one of two ways: Category view or List view. Category view displays the properties grouped together into relevant categories. List view lumps them all into one long list. The two icons at the top of the panel switch between these two views (Figure 2-16).

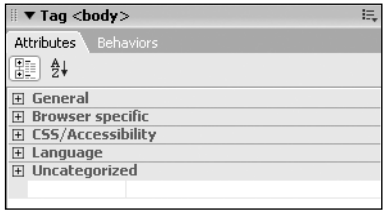


Figure 2-16. The Attributes panel in Category view

The Behaviors panel

The Behaviors panel lists the behaviors that have been applied (if any) to the selected tag (Figure 2-17). These should not be confused with server behaviors, which are pieces of code that run on the server before the page is served to the user.

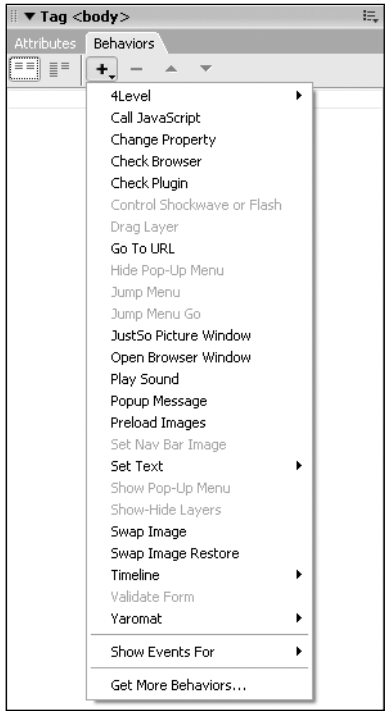


Figure 2-17. The Behaviors menu, which will grow as you install even more extensions!

Behaviors are usually JavaScript code that control or react to events on the web page that the user is interacting with, such as popping up windows and showing/hiding layers of the page. These types of actions are usually controlled by JavaScript; therefore, you use behaviors in Dreamweaver to apply that code to your pages.

The Files panel group

The Files panel group contains three panels by default: Files, Assets, and Snippets.

The Files panel

This panel is where you can access all the files of your site for editing and so forth (Figure 2-18). At the top of the panel are two drop-down lists. The left one lists all the sites you have currently defined in Dreamweaver. The bottom option in the list is Manage Sites, which launches the Manage Sites dialog, allowing you to add, edit, and delete sites, along with import and export site definitions and duplicate a site definition—all of which can speed up the site definition process quite a bit.

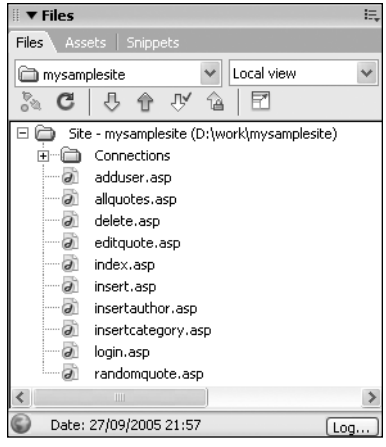


Figure 2-18. The Files panel showing local files

The drop-down list on the right allows you to choose which view you want to look at in the Files panel. The choices are Local view, Remote view, Testing server, and Map view. The first three options display the files of your site as per your site definition structure. If you haven't specified a remote server or a testing server for your site, these options will not display anything. If you have specified a remote server, the files at that location will appear. The Map view option will display a graphical representation of your website, so long as you have defined a homepage for your site.

Below the two lists is a series of icons that have some very useful functions. Left to right, they are as follows:

- Disconnect from remote host is grayed out unless you are connected.
- Refresh will refresh the listing of files displayed in the panel.
- Get file(s) will get the files currently selected in your site from the remote server and overwrite them if you tell it to.
- Put file(s) will put a copy of the files currently selected in your local site on the remote server.
- Check Out will check out the selected files, if you have check-in/check-out enabled for the site. Otherwise, this button is grayed out.
- Check In will check in the selected files, if you have check-in/check-out enabled for the site. Otherwise, this button is grayed out.

- Expand to show local and remote sites will replace the current Dreamweaver window with a full-size Files panel window that lists one set of files (local) on one side of the screen and another set of files (remote) on the other side of the screen (Figure 2-19). You can change what is displayed on the screen using the toggle buttons at the top of the screen, and you can return to the main Dreamweaver window by clicking the same icon (this time with the label Collapse to show only local or remote files) again.

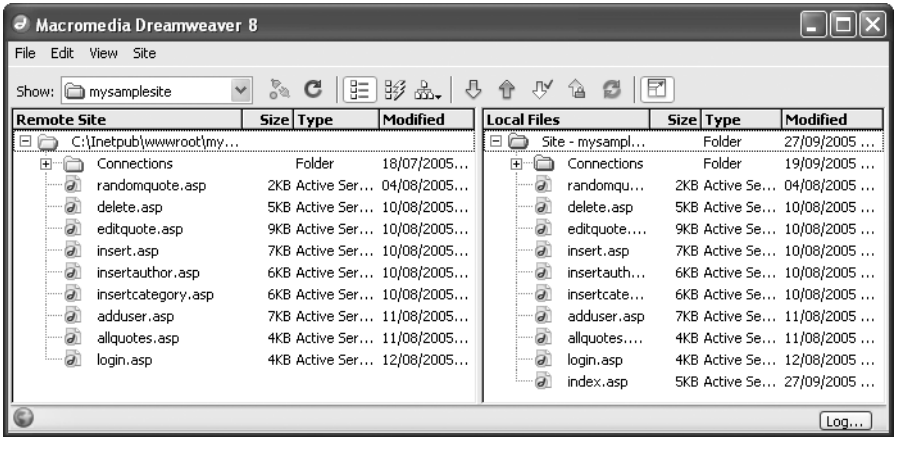

Macromedia Dreamweaver 8
File Edit View Site
Show: mysamplesite
Remote Site Size Type Modified
C:\Inetpub\wwwroot\my...
Connections Folder 18/07/2005...
randomquote.asp 2KB Active Ser... 04/08/2005...
delete.asp 5KB Active Ser... 10/08/2005...
editquote.asp 9KB Active Ser... 10/08/2005...
insert.asp 7KB Active Ser... 10/08/2005...
insertauthor.asp 6KB Active Ser... 10/08/2005...
insertcategory.asp 6KB Active Ser... 10/08/2005...
adduser.asp 7KB Active Ser... 11/08/2005...
allquotes.asp 4KB Active Ser... 11/08/2005...
login.asp 4KB Active Ser... 12/08/2005...
Local Files Size Type Modified
Site - mysampl... Folder 27/09/2005 ...
Connections Folder 19/09/2005 ...
randomqu... 2KB Active Se... 04/08/2005 ...
delete.asp 5KB Active Se... 10/08/2005 ...
editquote.... 9KB Active Se... 10/08/2005 ...
insert.asp 7KB Active Se... 10/08/2005 ...
insertauth... 6KB Active Se... 10/08/2005 ...
insertcate... 6KB Active Se... 10/08/2005 ...
adduser.asp 7KB Active Se... 11/08/2005 ...
allquotes.... 4KB Active Se... 11/08/2005 ...
login.asp 4KB Active Se... 12/08/2005 ...
index.asp 5KB Active Se... 27/09/2005 ...
Log...

Figure 2-19. The expanded Files panel displays local and remote files for easy management.

The Assets panel

Dreamweaver keeps track of all the media, or assets, that your site makes use of and categorizes them for you. The Assets panel is the place to look for your media when you need to add something to your page, be it an image, a Flash movie, a script, a template, or even a URL (Figure 2-20).

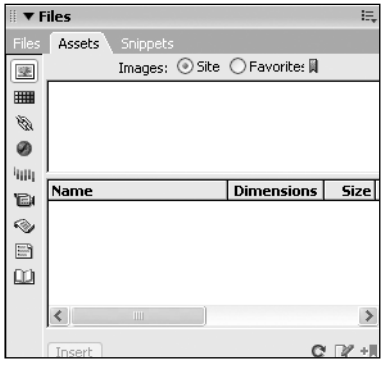


Figure 2-20. The Assets panel

The Snippets panel

Used properly, the Snippets panel can be a real productivity boon for web developers. The Snippets panel is a repository of code for you to reuse over and over again, saving you lots of time when it comes to doing the same thing on multiple sites (Figure 2-21). This panel comes populated with lots

of JavaScript code that covers many areas of JavaScripting, including populated drop-down menus (such as months and years), calculations (such as areas of circles, rectangles, etc.), conversion scripts (for time, weight, file size, etc.)—the list goes on and on. Check them out when you can. Some of them are quite useless, and some of them are outdated techniques, but the majority can be useful under the right circumstances.

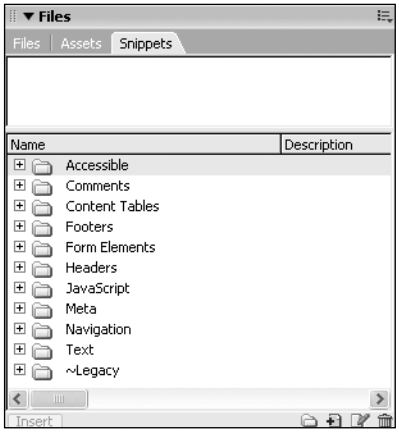


Figure 2-21. The Snippets panel contains lots of out-of-the-box code snippets.

The Properties panel

At the bottom of the Dreamweaver integrated development environment (IDE) sits the Properties panel, better known as the **Property inspector** or **PI** (Figure 2-22). This ever-changing panel displays the properties of whatever is selected in Design view. You can access the page properties by clicking the Page Properties button on the dialog when nothing is selected in Design view.

Figure 2-22. The Properties panel, also known as the Property inspector or PI

Your changes in Design view are immediately reflected in the Properties panel, where appropriate. If you make a change to your page in Code view, then the Properties panel needs to be manually updated before you can use it again to make property changes, and it will tell you the same (Figure 2-23).

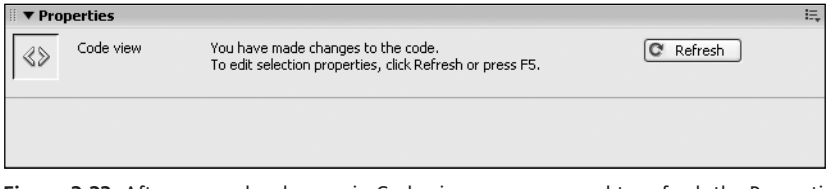


Figure 2-23. After you make changes in Code view, you may need to refresh the Properties panel.

The alternative to manually updating the Properties panel is to switch back to Design or Split view, which will also refresh the page.

The Properties panel is your friend, and you will undoubtedly use it a lot. We certainly make great use of it in this book!

The Results panel group

Sitting below the Properties panel is the Results panel group. It's an awkwardly named panel group due to its contents not all being "results" particularly, but we'll brush past that point. Of course, if its name really grieves you, you can always rename it to something you think more suitable for the job. Just right-click the panel name and select the Rename panel group option from the context menu.

The Results panel group contains eight panels by default (Figure 2-24): Search, Validation, Target Browser Check, Link Checker, Site Reports, FTP Log, Server Debug, and Reference.

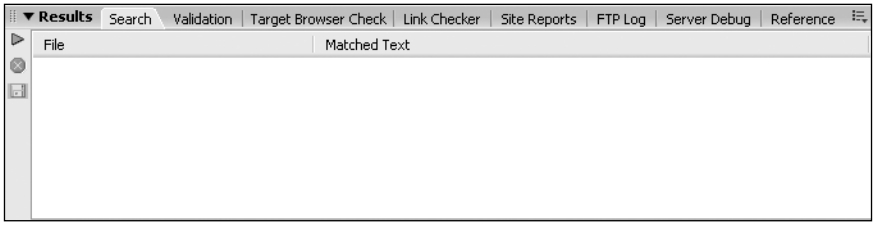


Figure 2-24. The Search panel in the Results panel group. Results of find and replace searches appear here.

The icons down the left side of the report area change from panel to panel, but some remain the same, such as Save report, which does exactly that.

The Search panel

The Search panel is actually a results panel for find and replace operations that are performed. Each result line is clickable and will take you to that location of the document it refers to, to show you what was replaced, or found. If you have purchased a third-party file comparison tool, such as the one listed on the Macromedia website and made by Araxis, then you can tell Dreamweaver to make use of it via the Preferences section (Edit ➤ Preferences ➤ File Compare). After you specify a tool to compare files, the Compare files option becomes usable in Dreamweaver; you can find this option in the context menu (right-click) in the Files panel. Results of the comparison tool's findings are displayed in the Results panel.

The Validation panel

The Validation panel displays validation errors encountered on the page, when the validator is run. The settings for the validator are in the Preferences area, which you can access by selecting Edit ➤ Preferences and then choosing Validator, or by clicking the green arrow and choosing the Settings option.

The Target Browser Check panel

In the Target Browser Check panel, you can run checks against the list of browsers you've specified that you want your site or page to work in. Again, you can select the options for this panel by choosing Settings from the menu that pops up when you click the green arrow. The drop-down list at the top of this dialog allows you to select whether you want to check the current document only or check the entire site.

The Link Checker panel

The Link Checker panel offers three options. You can check your website for broken links, external links (or links that go to pages not contained in your local website), and orphaned links. These options can be run only once an entire site scan has been performed, so Dreamweaver has a site cache that it can use to check for orphaned links, which are links to files in your local site that no longer exist.

The Site Reports panel

The Site Reports panel displays the results of the report you ask it to run. To run a report, click the green arrow and the Reports dialog will appear. Here you can specify what you want to see a report on. Just select the items relevant to your report and click Run (Figure 2-25).

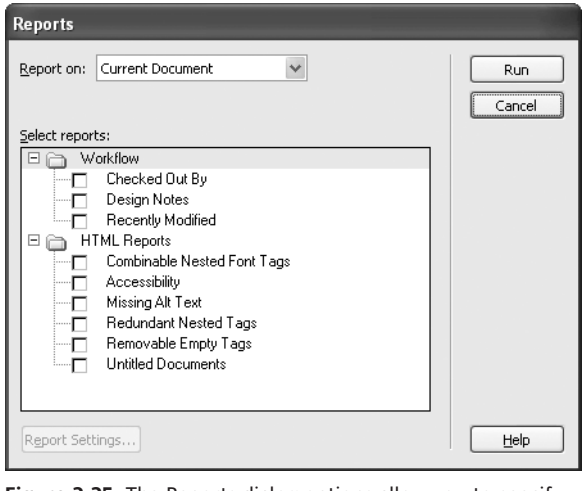


Figure 2-25. The Reports dialog options allow you to specify what level of detail you want to see in your report.

Once the report is complete, the results are displayed in the list. Be aware that this is not a flawless mechanism and should not be relied upon solely—get a second opinion!

The FTP Log panel

The FTP Log panel displays a log of all the FTP activity that has been processed in Dreamweaver. If you use Dreamweaver for your FTP access (and it has improved remarkably since the last incarnation), then you can check all FTP activity here.

The Server Debug panel

The Server Debug panel is useless for ASP VBScript sites and can be ignored.

The Reference panel

The Reference panel has moved from the side panels down to the much wider bottom panel, so reading its content is now a much more comfortable experience than in previous Dreamweaver versions.

Thirteen reference books are available in this panel. These books cover a wide range of topics, many of which web developers today need to have a grasp of, including HTML, JavaScript, CSS, XML, XSLT, SQL, and more.

Toolbars

The toolbars sit at the top of a document by default. Some developers like to have them docked as part of the IDE, and remove them from a document and dock them below the Insert toolbar. Choose the location for them that works best and is most comfortable for you.

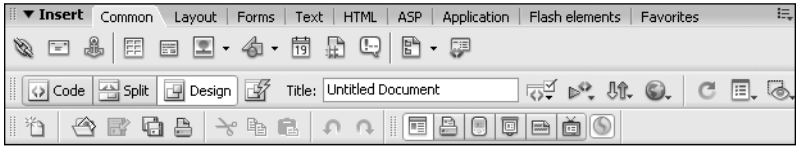


Figure 2-26. The Insert bar in Tab mode, along with the other toolbars available to the workspace

The Insert toolbar/panel group

The Insert panel group is a kind of crossbreed. It's not really a panel group like the other the panel groups, but it does contain lots of panels, so we'll stick with the analogy for now.

You can view the Insert panel in Menu mode as shown in Figure 2-27 or in Tab mode as shown at the top of Figure 2-26.

Figure 2-27. The Insert toolbar in Menu mode

Each of the panels in the Insert bar contains shortcut icons to perform an action that can also be performed by selecting one of the main menu selections found on the Insert menu.

There is also a tab called Favorites into which you can assemble your own custom-built toolbar that provides your most used Insert icons on one handy bar. To begin building your Favorites bar, select the Favorites tab, and then right-click in an empty part of the bar. Select Customize Favorites to launch the Customize Favorite Objects dialog (Figure 2-28).

Figure 2-28. The Customize Favorite Objects dialog, where you can create your own handy toolbar of regularly used objects

You can select the element you want to appear on your Favorites toolbar in the left list, click the double arrow button in the middle, and that element will appear in the list to the right. If you need to add a separator to the bar, click the Add separator button at the bottom of the dialog. You can also reorder the items in the list by selecting an item to move and then clicking the up or down arrow to move it in that direction. To remove an item from the bar, click the trash can icon above the list on the right after you select the item(s) to remove.

Once you're happy with the contents of your Favorites toolbar, click OK and the toolbar will be built for you (Figure 2-29 shows an example).

Figure 2-29. A customized Favorites toolbar

You can, of course, edit the content of the Favorites bar at any time by right-clicking an empty area of the bar and selecting Customize Favorites again.

The Document toolbar

The Document toolbar contains several buttons and one input text field. The text field is for editing the title of the page you are working on (Figure 2-30).

Figure 2-30. The Document toolbar

The three leftmost buttons, Code, Split, and Design, allow you to switch between the three views available in Dreamweaver. The remaining buttons, from left to right, are as follows:

- Live Data View: This button is a very powerful feature that allows you to view actual data from your database on the page you are creating, while you are creating it. You can make design changes (or code changes) to your page and see—in real time—what presentational effect they will have on the data being displayed. (The page's Title field separates this button from the rest of the buttons.)
- Browser Check: This button gives you quick access to run a browser check and to the settings for what you would like to have checked. You can also specify if the browser check should run when you open a page or not, and you'll find links to the relevant report page in the Results panel to view any errors that it finds (Figure 2-31). This toolbar button will change to let you know if the page passed or failed the checks.

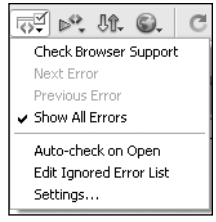


Figure 2-31. The Browser Check menu options

- Validate Markup: This button pops up a menu that gives you options to check the markup of the current document, the entire site, and selected files in the site, and also to specify the settings that you would like to validate against (Figure 2-32).

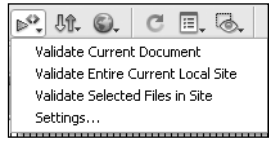


Figure 2-32. The Validate Markup menu options

- File Management: This button is used for quick access to file actions such as `Get`, `Put`, and `Locate`. The Locate in site feature is useful if you are editing lots of files at once in a large site and need a quick reminder which file you are looking at (we all create lots of `index.asp` pages, right?). Clicking Locate in site will expand the Files panel to show the currently file selected in the list.
- Preview/Debug in Browser: This button allows you to launch one of the browsers listed for preview/debugging purposes. You can also edit the browser list from this menu.
- Refresh Design View: This button enables you to bring Design view into line with any changes you may have made in Code view. Most of the changes you make will be reflected in Design view, but for those times when they are not, clicking this button does the trick.
- View Options: This button lets you decide how you want certain visual elements to be displayed (Figure 2-33). You can use any or all of the visual aids that Dreamweaver has to offer, and you use this menu to specify which ones. If you work in Split view, you also have the option of setting Design view to be above or below the code window. The menu contains different options in Code view that are relevant to code editing only.

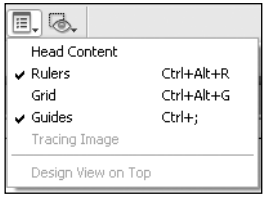


Figure 2-33. The View Options menu options to quickly show and hide Design view layout helpers

- Visual Aids: This button gives you several options for turning on or off visual items that Dreamweaver can display in Design view to help you while developing your web pages. With Dreamweaver's vastly improved CSS handling, the new additions to this menu include showing CSS layout backgrounds, the CSS layout box model, and CSS layout outlines (Figure 2-34). This button is disabled in Code view (obviously!).

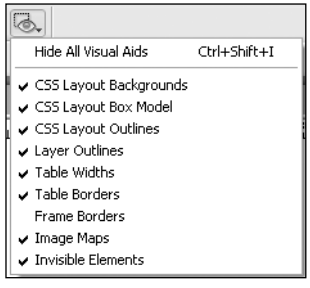


Figure 2-34. The Visual Aids menu options, now with superb CSS handling

The Standard toolbar

This toolbar gives you quick access to create new files, open existing files, save an open file (if it needs to be saved), and save all open files. The latest addition to this toolbar is the Print Code button. The rest of the toolbar is pretty standard (pun intended!), with Cut, Copy, and Paste buttons, along with Undo and Redo (Figure 2-35).

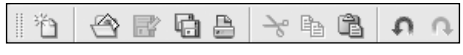


Figure 2-35. The Standard toolbar

If you need to set a different number of allowed undos in Dreamweaver than the default of 50, select Edit ➤ Preferences, and in the General section, change the value in the Maximum number of history steps text field. The larger the number of undos, the more memory Dreamweaver will use to remember what happens as you use it, so be judicious!

The Style Rendering toolbar

The Style Rendering toolbar is hidden by default, but you can easily unhide it by right-clicking anywhere in the toolbar area and selecting the Style Rendering option (Figure 2-36).

Figure 2-36. The Style Rendering toolbar

This toolbar is a new addition to Dreamweaver, and a welcome one at that. With a good deal of the focus of this release on CSS enhancements, the Style Rendering toolbar is a great benefit for those who create websites that will be viewed on different mediums, such as WebTV or handheld devices. If you create separate stylesheets for these specific mediums, you can see what the content will look like rendered in Dreamweaver's Design view using this toolbar. Click the appropriate button on the toolbar and Dreamweaver will render your page using the stylesheet you specified for that medium.

The options on the Style Rendering toolbar as are follows:

- Screen: View the page as it would appear on a computer screen.
- Print: View the page as it would appear printed.
- Handheld: View the page as it would appear on a handheld device, such as a mobile phone or PDA.
- Projection: View the page as it would appear on a projection device.
- TTY: View the page as it would appear on a teletype machine.
- TV: View the page as it would appear on a television screen.
- Toggle CSS: Turn CSS rendering on or off in Design view.

The Coding toolbar

There have been some great improvements to the coding environment in Dreamweaver. A massive boost for productivity here is the long-awaited Coding toolbar (Figure 2-37).

Figure 2-37. The vertical Coding toolbar, which can be seen only in Code view

The Coding toolbar provides quick access to many standard coding operations, such as highlighting invalid code, applying and removing comments, indenting and outdenting code, and inserting recently used code snippets. One of our favorite features of the toolbar is the ability to collapse and expand code selections. This is a superb feature, and once you've used it for a while you'll wonder how you ever did without it.

To collapse a section of code, follow these steps:

1. Highlight every line that you want to include in the collapsed section, and then click the Collapse button. The easiest way to select lines of code may be to click the line numbers if they are visible; otherwise, click in the thin blue bar where they appear, between the Coding toolbar and the code window.
2. Click the line where you want to begin the collapse and drag down to the end of the section to collapse. As you drag, a line is drawn with a box at each end with a minus symbol in it.
3. When you release the mouse at the endpoint of the section you want to collapse, the line stays in place.

Click the minus box icon to collapse that section, which leaves just a few characters visible in a gray/brown box for identifying the section collapsed. Click the plus sign button to expand the area with again. Clicking in the document away from the collapsed selection will hide the plus sign icon. Clicking a collapsed section's gray/brown box will display the plus sign icon for you to expand the selection again.

The Expand All toolbar button will do exactly that: expand all collapsed code blocks. Bear in mind that once you expand an item, if you don't collapse it again before clicking elsewhere in the document or otherwise moving the selection focus to something else, the selection for the code collapse will disappear, and to collapse it again you will need to reselect the code again first. While the code block remains selected, the collapse area remains active. If you expand an area to edit it, you will need to reselect the area before you can collapse it again. It's a bit of a bind, but that's the way it works. It's a great tool for helping to navigate those long pages of code.

The Coding toolbar is visible only in Code view and appears on the left side of the code window. You can't undock it, but you can hide it if you don't want to use it.

New tools in Design view

Design view now features a couple of handy new tools: a Zoom tool, a Magnification list, a Hand tool, and a Guides feature. With the exception of the Guides feature, these items can be found at the bottom of the document window, to the right of the tag selector bar (Figure 2-38).

Figure 2-38. New Design view tools for zooming and panning

Zoom

The magnifying glass icon is the Zoom tool. Clicking this button allows you to drag an area of the page to zoom in to, or to simply click to zoom in by one magnification step at a time. Pressing the Alt key when clicking zooms back out one level at a time.

You can now get really close to the page to fine-tune anything you might be having trouble with, such as alignment of elements on the page. Zooming right in might help you to resolve these types of issues.

Magnification

The Magnification list has a preset selection of zoom levels for you to choose from. Selecting one of them will zoom to that level in the open document, starting at the top left of the page—unless you were already zoomed on a specific area, in which case it will continue to zoom on that area.

Hand

The Hand tool enables you to pan around the document when you are zoomed in to it. To return to editing mode to modify the content on the page when zoomed in, click the black arrow icon.

Guides

The Guides feature goes hand in hand with the ability to zoom and pan. Guides can help you lay out your documents with pixel-perfect precision.

Select View ➤ Guides to display a menu with several options for controlling how guides should be seen (or not seen) in the Dreamweaver workspace (Figure 2-39).

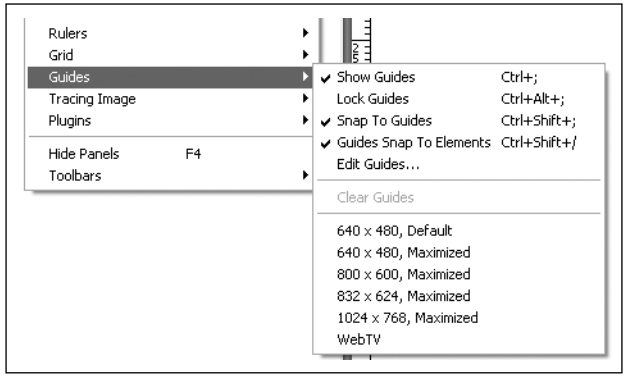


Figure 2-39. The Guides menu, along with several preset standard website sizes

The lower portion of the menu offers preset guide sizes which, when selected, will add guides at those sizes to the document window.

You can also access the Guides settings by selecting View ➤ Guides ➤ Edit Guides (Figure 2-40).

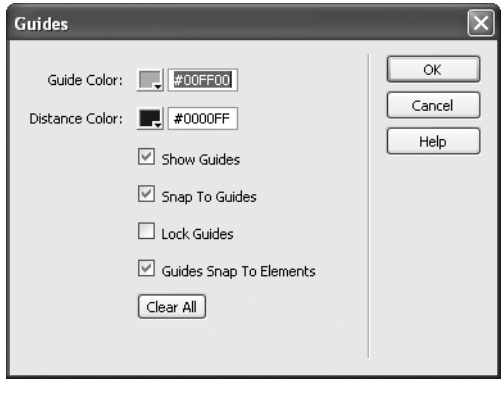


Figure 2-40. Editing guide properties in the Guides dialog

If you have the rulers turned on (View ➤ Rulers ➤ Show), you can drag guides out from the top or the side of the document by clicking the ruler and then dragging. A guide is created and a little yellow tooltip tells you the pixel position as you drag it along. The tooltip also appears when you mouse over a guide that is already positioned in the document (Figure 2-41).

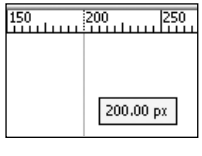


Figure 2-41. A guide being positioned at 200 pixels across the page

Defining an ASP VBScript site in Dreamweaver

Now that you're familiar with the Dreamweaver environment and basic features, in this final section of the chapter, we'll define an example site in Dreamweaver. To create a site in Dreamweaver, you use the New Site dialog box, which actually has several screens within it. To launch the New Site dialog, select Site ➤ New or choose Manage Sites from the bottom of the Sites list in the Files panel, and then click New.

You should be presented with a Site Definition dialog box and, if this is the first time you have run this wizard, you will likely be presented with the Basic option. In this section, we will run through the Advanced dialogs, so click the Advanced tab to switch views before continuing. This dialog box has eight screens, but we will only be covering the first three here, as this is enough to get a basic site set up.

Local Info screen

The Local Info screen looks like Figure 2-42.

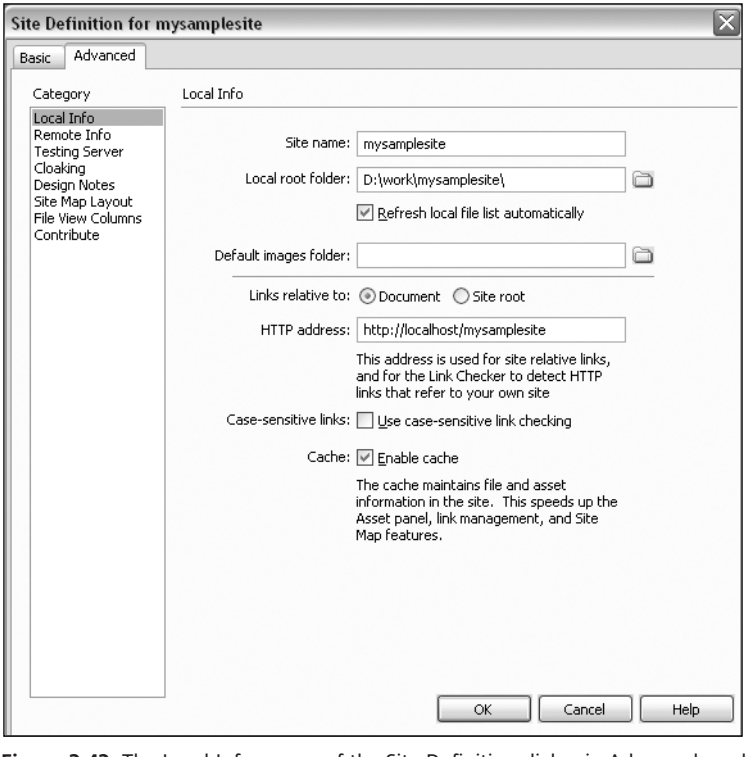


Figure 2-42. The Local Info screen of the Site Definition dialog in Advanced mode

Fill it in as follows:

- Site name: Name your site mysamplesite.
- Local root folder: Browse for the folder where site files are kept on your computer. There may not be any as yet, but the root folder needs to be specified here.
- Default images folder: This field can be left blank.
- HTTP address: Type in http://localhost/mysamplesite.

Also, make sure the site cache is enabled (i.e., select the Enable cache check box).

If you want to enforce case-sensitive link checking, check the Use case-sensitive link checking box and Dreamweaver will make sure that when you link to a file called `index.asp`, for example, it isn't linking to `Index.asp` if case sensitivity matters on your web hosting server. On most it doesn't, but be aware that on some it does (such as Linux).

Remote Info screen

Switch to the Remote Info screen, which looks like Figure 2-43.

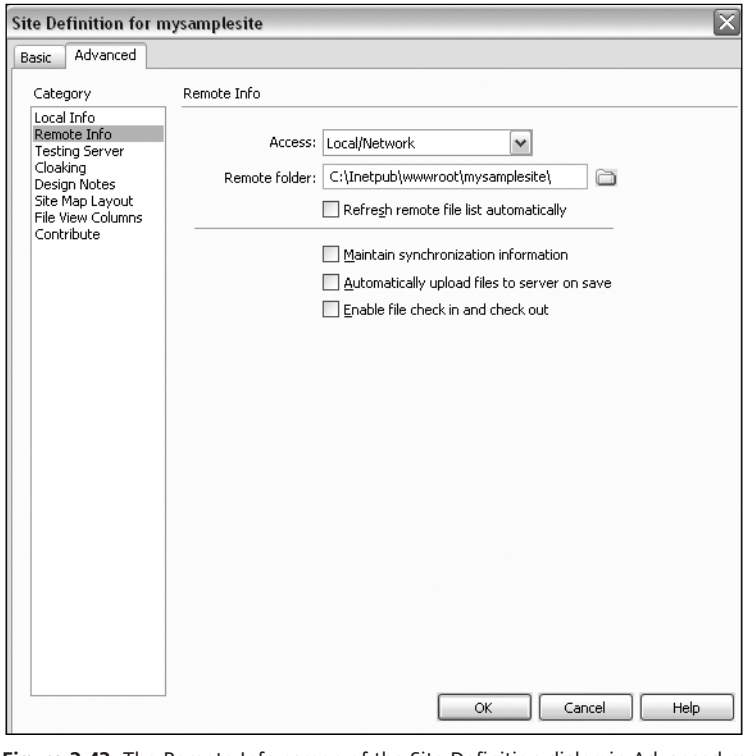


Figure 2-43. The Remote Info screen of the Site Definition dialog in Advanced mode

Select the following options:

- Access: Select Local/Network.
- Remote folder: Browse for the location of the remote folder. It's usually in the inetpub\wwwroot folder on your C drive.

You can leave the other settings at their defaults. Refresh remote file list automatically is selected, which means that when you view the remote file list in the site window, it will make sure you are seeing the latest version of the remote files. Also, you don't need to select Enable file check in and check out; this option is useful for collaborative environments, as it allows users to quickly see who is working on what file and helps to prevent work being overwritten when two people save the same file.

Testing Server screen

Switch to the Testing Server screen, which looks like Figure 2-44.

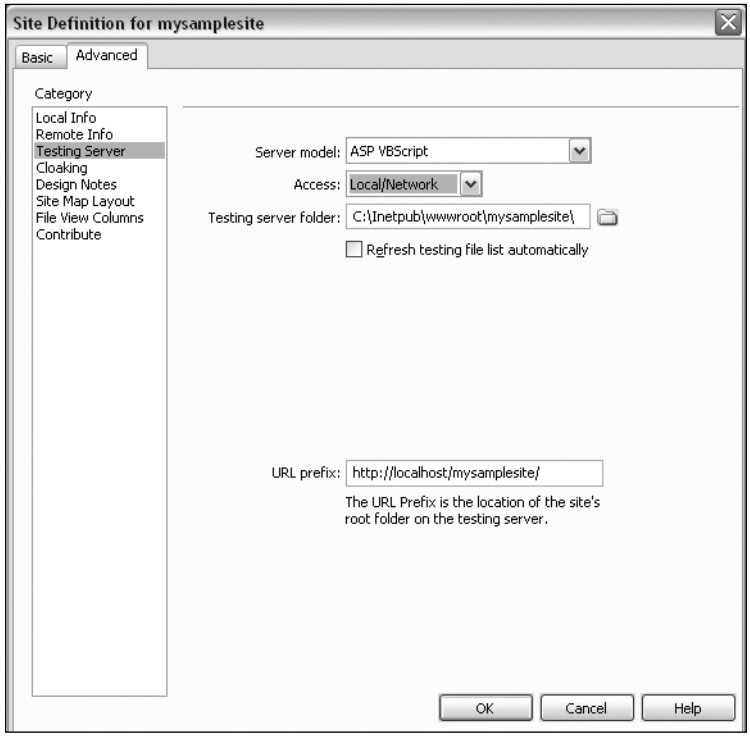


Figure 2-44. The Testing Server screen of the Site Definition dialog in Advanced mode

Select the following options:

- Server model: Select ASP VBScript.
- Access: Select Local/Network.
- Testing server folder: Browse for the testing folder, which is where Dreamweaver will go to find the file it should run when you are testing. This can be set to the same name as the remote folder in a simple development environment.
- URL prefix: Type http://localhost/mysamplesite and click OK.

Creating the site

Let's recap briefly what you've done so far.

- You created a physical directory called mysamplesite on your PC, where you will save all your site files.
- You created a virtual directory named mysamplesite that will be accessed from the browser from `http://localhost/mysamplesite`. This will lead to the root of your site.
- You defined a site in Dreamweaver named mysamplesite that will be tested on your local server.

With the preceding three screens of the Site Definition dialog completed, click OK to create the site. If you began the creation of the site from the Manage Sites dialog, you will be returned to that dialog now and your new site will be listed there (Figure 2-45).

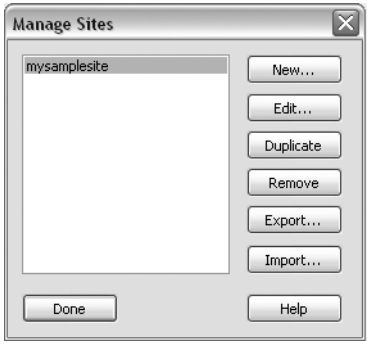


Figure 2-45. The Manage Sites dialog listing the newly created site

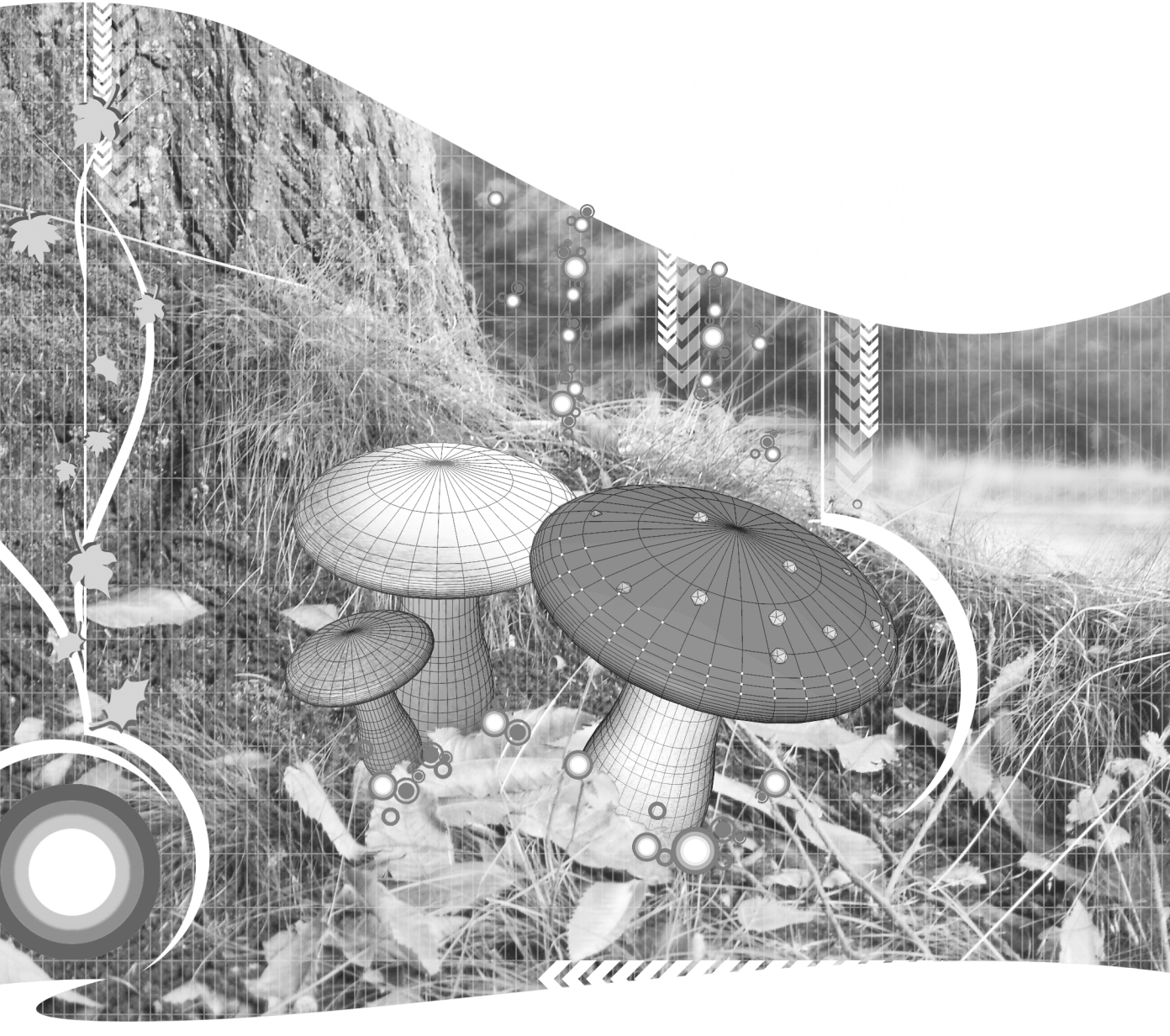
If this dialog is open, click Done to close it. The Files panel will update to show your new site.

If you did not begin creation of the site from the Manage Sites dialog, your site will simply be selected automatically in the Files panel when you click OK to create it from the New Site dialog.

Conclusion

The new release of Dreamweaver is heavily focused on CSS, with some updates and new additions to the coding environment and overall capabilities of the application as a whole.

In this chapter, you explored the interface to see what's new and what's changed from the previous version. We have hopefully covered enough of the interface for you to be able to quickly access the items you need at any time. You also walked through the basic steps involved in getting a site set up in Dreamweaver, ready for building a dynamic ASP VBScript website.



Chapter 3

A FIRST TASTE OF ASP

You must be thinking to yourself (or even out loud, if you're like me), "How do I integrate ASP code into my Dreamweaver pages?" You've gone through a nice general overview in Chapter 1 about ASP and how it can turn your static site into a dynamic one, but you want more details. How does it look? How hard or easy is it to code? How exactly can I accomplish those dynamic features with ASP?

Although Dreamweaver does a great job of generating code that works via its server behaviors, if an error occurs, or if you want to customize your code further, then you need to know how the underlying code actually works. That's why this chapter is here—to teach you the basics of ASP and VBScript. In this chapter, you'll learn about the following:

- Syntax rules
- Variables
- String, integer, and Boolean data types
- Concatenating code
- Commenting your code
- VBScript built-in functions
- Operators
- Conditional logic

- Looping logic
- Cookies
- Session variables
- Environment variables

Specify your language: VBScript

The first thing you need to do is specify the scripting language you'll be using with ASP. By now, you know that you'll have Dreamweaver script your ASP pages in VBScript, not JavaScript.

Thankfully, Dreamweaver inserts the appropriate ASP **directive** to specify the language, which saves a lot of time. You don't have to write out this line of code on each page. In any case, without the ASP directive, ASP defaults to VBScript, not JavaScript—so it's even more important to specify the language if you're scripting ASP in JavaScript. For you, though, specifying the language is simply good practice.

1. Load Dreamweaver, and create a blank dynamic ASP web page by selecting File ➤ New, highlighting Dynamic page under the Category list of the General tab, highlighting ASP VBScript under the Dynamic page list, and pressing OK.
2. Switch to Code view, and note the first line:

```
<%@LANGUAGE="VBSCRIPT" CODEPAGE="1252"%>
```

This is the ASP directive that specifies the scripting language. This line must be the first line of your code, because, as mentioned, you must tell ASP which language you'll be using *before* you start coding. If not, ASP will throw an error on your page. It will throw an error in two cases:

- If you don't specify a language in this directive, and you code in any language other than VBScript (because the server assumes VBScript by default)
- If you code in a language other than the one you specify

ASP delimiters <% %>

As you learned in the first chapter, all ASP code must be written within ASP delimiters, like so:

```
<% Response.Write "Hello World" %>
```

This is also OK:

```
<%
Response.Write "Hello World"
%>
```

ASP will ignore the empty spaces between the opening delimiter and the beginning of your code line, and between the end of your code line and the closing delimiter. A pair of open and closed delimiters is considered to be an ASP **block**. These delimiters let the server know that ASP code is coming, so that it knows to parse the code properly (and not think that it's HTML, PHP, or any other language).

You can close and reopen new delimiters for each line of ASP code, but it's better practice to keep related lines of ASP code in one block. For example, look at the following two code snippets, which show the same code written in one and two blocks.

Here's the one-code-block version:

```
<%
Response.Write "Omar"
Response.Write "Elbaga"
%>
```

And here's the same code divided into two code blocks:

```
<%
Response.Write "Omar"
%>
<%
Response.Write "Elbaga"
%>
```

The style is up to you, but always keep your code consistent. Remember that it's always better to group similar lines of ASP code into one code block, but it's possible to break up lines into multiple code blocks. Sometimes this is actually necessary, since HTML code often must be surrounded by ASP code, as you'll see later in this chapter.

Keep in mind that a single line of ASP code can't be broken up into multiple blocks; you can only break up multiple lines into multiple blocks. For example, the following is illegal, because it's a single line of code and needs to be written as one line:

```
<%
Response.Write
%>
<%
"Omar Elbaga"
%>
```

Insensitive case!

Being insensitive is usually considered distasteful to people; however, in the case of your scripting language, it can be really helpful. **Case insensitivity** means that the code doesn't really care about whether you write your VBScript in uppercase or lowercase. For example

```
<%
Dim myname = " Strawberries"
%>
```

is the same as

```
<%
DIm mYnAme = " Strawberries"
%>
```

This is not the case (no pun intended) with languages such as JavaScript, C#, Java, and others. When a language is case-sensitive, it will throw errors if you use the wrong case for a function name, or if you declare a variable in lowercase and then use it in uppercase later on in the code. In a case-sensitive language, `pizza` and `PIZZA` are considered to be two different variables.

Hardcore coders like this idea of case-sensitivity, because to them it's neater, more organized, and keeps you in check as a coder, which is what programming is about. However, for the beginner, it can be easier to learn a case-insensitive language—it's one less thing to worry about.

> *Even though VBScript is case-insensitive, try to stay in the habit of keeping the case of your code consistent. Remember that consistency is key to all good coding, not only for you but for others, because it makes the code easier to read and therefore easier to debug when you come up against errors. Here's an obvious example:*
>
> *This is easy to read. This is easy to read. This is easy to read.*
>
> *This is hardEr to read. ThiS IS hArdEr to ReAd. tHIs is HaRdEr to rEad.*

Variables

The variable is one of the most important aspects of a programming language. A variable is a sort of placeholder for any value that's likely to change. For example, take a login page where many different users will log in to a web application. The user ID and password will be different for each user, so instead of referencing a literal (meaning an actual) value, you store the values the user enters as variables, and then check these variables against the information stored in a "login details" database to see if they're correct.

Variables capture the essence of what I mean by dynamic—a single page can be used to log in hundreds of site visitors. A variable's value can change, so you don't have to worry so much about the actual values while you're coding—unless you'll be manually specifying the values of a variable, which is rarely the case. The values will usually be retrieved from your site visitors and used immediately, or simply stored separately in a database and called, changed, or deleted when needed. In this book, however, you'll start off by creating examples for which you manually specify the value of the variables just to become familiar with the way that they're created and changed. In future chapters, you'll move on to allowing site visitors and databases to create the values for variables.

Now that you understand what variables are, let's look at how to create them, and what they look like in code.

Declaring and printing variables

Let's get a little more specific on how to actually create variables in ASP VBScript. Remember, all ASP code must be within what? Delimiters! You got it.

1. Create a new dynamic ASP VBScript page, as before.
2. Switch to Code view and enter the following code after the opening <body> tag (see Figure 3-1):

```
<%
  Dim myUsername
  myUsername = "Omar"
%>
```

```
1  <%@LANGUAGE="VBSCRIPT" CODEPAGE="1252"%>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml">
4  <head>
5  <meta http-equiv="Content-Type" content="text/html;
   charset=iso-8859-1" />
6  <title>Untitled Document</title>
7  </head>
8
9  <body>
10 <%
11   Dim myUsername
12   myUsername = "Omar"
13 %>
14
15 </body>
16 </html>
```

Figure 3-1. In Code view, the code that declares the variable myUsername

In the first line, you're using the `Dim` keyword to explicitly declare the variable `myUsername` (as opposed to declaring it implicitly, which will be discussed shortly). In the second line, you're giving the variable its value.

3. Now load the page in your web browser. What do you see? Nothing? Well, that's great—that's what you're supposed to see. Can you guess why? Because all you did was declare a variable. You didn't tell ASP what to do with the variable. You can do many things with a variable—for example, you can enter it into a database, perform a mathematical function, and then pass it from page to page within an application.
4. Now add the following new bold line of code to your previous code, and view the page again (see Figure 3-2):

```
<%
  Dim myUsername
  myUsername = "Omar"
  Response.Write myUsername
%>
```

```
1  <%@LANGUAGE="VBSCRIPT" CODEPAGE="1252"%>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml">
4  <head>
5  <meta http-equiv="Content-Type" content="text/html;
   charset=iso-8859-1" />
6  <title>Untitled Document</title>
7  </head>
8
9  <body>
10 <%
11   Dim myUsername
12   myUsername = "Omar"
13   Response.Write myUsername
14 %>
15 </body>
16 </html>
```

Figure 3-2. In Code view, the code that declares and prints the variable myUsername

In the preceding code, you've used `Response.Write` to write out a variable. Notice that the web browser now displays the value of the variable (see Figure 3-3).

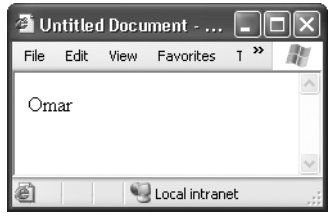


Figure 3-3. The value of the variable myUsername displayed in the web browser

5. Next, you'll use the <%= shortcut from Chapter 1 to write it out in a different way. Change the code you've added so far to the following:

```
<%
  Dim myUsername
  myUsername = "Omar"
%>
Hello <%=myUsername %>
```

6. Now save the page and view it in your browser again (see Figure 3-4). You should see the following text appear:

   Hello Omar

Figure 3-4. Output of the code using the ASP shortcut (<%=) in the web browser

In this example, you, the web developer, declared the value of the variable labeled `myUsername`. Imagine if you allowed the web user to set the value of the variable. Notice how you could greet any web user without actually knowing his or her name! This is how websites do it. In many cases, the value of the variable is simply pulled from a database, which you'll learn more about in chapters to come.

Let's do another quick example using the same code, but for which the web surfer specifies the name. Create a new dynamic ASP VBScript page and save it as `sendnameback.asp`.

In Code view, type the following code:

```
<%
  Dim myUsername
  myUsername = Request.Form("txtname")
%>
Hello <%=myUsername %>
```

Switch to Design view, and press Enter to create a space between the greeting and the form you're about to build. From the standard toolbar, insert a form. With the cursor inside the form, select Insert ➤ Form ➤ Text Field. Name the text field txtname. Place the cursor to the right of the text field and select Insert ➤ Form ➤ Button. Highlight the form, and from the Properties panel, set the Action to go to the same page. Also, make sure the Method is set to "POST". You will learn more about this in Chapter 5. Load the page in your browser (see Figures 3-5 and 3-6).

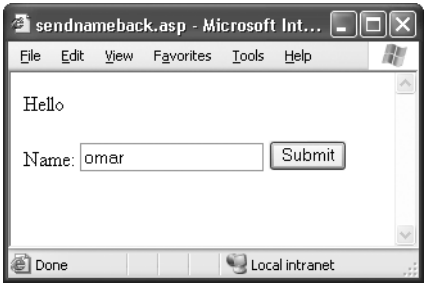


Figure 3-5. View of sendnameback.asp as the name is being submitted

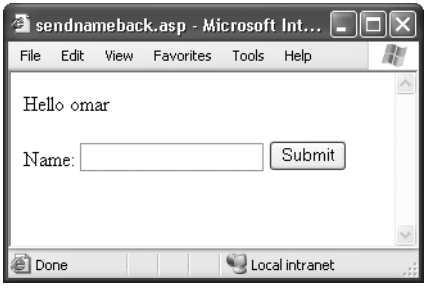


Figure 3-6. View of sendnameback.asp in the web browser after the name has been submitted

In this code, you set the value of the variable named myUsername to become whatever the user submits in the text field named txtname. The variable is then simply displayed upon submission of the form. This is an example of how the values for variables change dynamically, because whatever is input in the form will become the value of the variable named myUsername. You could also have this form inserted into a database, which you'll learn about in Chapter 4.

Bear in mind that ASP code is processed in the order that it appears on the page. As such, you must declare a variable and give it its value before attempting to print it. For example, the following code won't work:

```
Hello <%=myUsername %>
<%
Dim myUsername
myUsername = "Omar"
%>
```

Because ASP doesn't know what myUsername *equals until you give it a value, the variable will just be empty when this page is displayed. In a way, ASP is forgiving in that a variable that hasn't been given a value will stay empty upon display, instead of throwing off the entire web application; other languages may throw errors in such cases.*

Letting VBScript declare variables implicitly

You can also give a variable its value and allow VBScript to automatically declare the variable implicitly, as follows:

```
<%
myUsername = "Omar"
%>
```

Compare the preceding implicit declaration with the following explicit one:

```
<%
Dim myUsername
myUsername = "Omar"
%>
```

> *It's time-saving to let VBScript declare the variable implicitly, but it's better to declare a variable explicitly before giving it its value, because there's an ASP directive,* `<% Option Explicit %>`*, that can be added at the top of the page that will* force *you to declare variables explicitly. Dreamweaver doesn't use this directive, so you won't have any problems if you don't explicitly declare your variables, but it's good to know this for future reference, and it's good practice.*

Syntax rules for variables

As with all languages, there are some rules that you can't break when declaring variables in VBScript. Here are the ones you need to know when declaring the name of your variable:

- Don't use spaces, periods, or hyphens. `My Variable`, `My.Variable`, and `My-Variable` are all unacceptable.
- Variables must begin with a letter, not a number. `2myVariable` is unacceptable.
- Reserved words can't be used as names for your variables, because, well, they're reserved. For example, you can't use `Dim` as the name of your variable because that stands for something in VBScript already. For a list of VBScript reserved words, see the following link:

  `http://support.microsoft.com/default.aspx?scid=kb;EN-US;q216528`

Data types: string, integer, and Boolean

Every variable has a specific data type. This is important to know, because each data type has its own function, and you have to know when to use one data type rather than another. This section will cover three of the most important data types: string, integer, and Boolean. The following includes these as well as list some of the other data types you may come across in your programming future:

- **String**: used for any alphanumeric character that can be up to about 2 billion characters in length
- **Integer**: used for an integer ranging from –32,768 to 32,767
- **Boolean**: used to hold the values `True` (`-1`) or `False` (`0`)
- **Empty**: used for a variable that hasn't been initialized. An empty numerical variable will have a value of `0`, and an empty string variable will have a zero-length string (`""`)
- **Null**: used when a variable has no valid data
- **Byte**: used for an integer in the range of 0 to 255
- **Currency**: used for a number between –922,337,203,685,477.5808 and 922,337,203,685,477.5807
- **Long**: used for an integer ranging from –2,147,483,648 to 2,147,483,647
- **Single**: used for a number ranging from –3.402823E38 to –1.401298E-45 for negative values, and 1.401298E-45 to 3.402823E38 for positive values
- **Double**: used for a number ranging from –.79769313486232E308 to –4.94065645841247E-324 for negative values, and 4.94065645841247E-324 to 1.79769313486232E308 for positive values
- **Date (time)**: used to express the date and/or time; uses a number that represents a date between January 1, 100 and December 31, 9999

Again, the three most important data types, which you'll be looking at in detail here, are string, integer, and Boolean.

A data type is also referred to as a variable type. It's importance is that a variable will always act according to its type. You won't get the same results if you do the same thing with a string that you do with an integer. Strings contain alphanumeric characters, and integers contains numerical digits. You'll need to remember that strings must *always* be surrounded by quotation marks, but integers never do. Booleans hold the values `True` or `False`. The numerical value for `True` is `-1` and the numerical value for `False` is `0`.

Let's look at an example to demonstrate the different variable types. In Dreamweaver, switch to Code view, create a new ASP page named `datatypes.asp`, and enter the following between the `<body></body>` tags:

```
<%
  myString = "Hi, my name is Omar! I am "
  myInteger = 26
  myBoolean = True
  ' notice how only strings are surrounded by quotes.

  Response.Write myString & myInteger & " That is " & myBoolean
%>
```

View this in your web browser, and you should see the output shown in Figure 3-7.

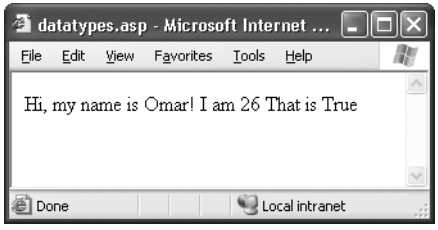

datatypes.asp - Microsoft Internet ...
File Edit View Favorites Tools Help
Hi, my name is Omar! I am 26 That is True
Done
Local intranet

Figure 3-7. View of datatypes.asp in the web browser after the ASP is code is processed

As opposed to some other programming languages, with ASP VBScript, you don't specify the data type. The VBScript processor on the server picks it up on its own. Remember that when setting the value of a variable, if you surround a number with quotes, the ASP processor will see it as a string and not an integer—which is OK if that's what the scenario calls for, such as numbers appearing amidst text. Numbers need to be of an integer data type if you'll be performing mathematical computations with them.

Variable concatenation

Concatenation is a fancy word for "joining things together." It doesn't mean adding things together mathematically, it simply means putting things together side by side. In VBScript, the ampersand (&) is the operator that concatenates strings.

As an example, let's try concatenating two strings into one variable. Create the following file and save it as `my_name.asp`. Enter the following code within the <body> tag while in Code view:

```
<%
  myFName = "Omar"
  myLName = "Elbaga"

  Response.Write myFName & " " & myLName
%>
```

View the page in your browser, and you should see the output shown in Figure 3-8.

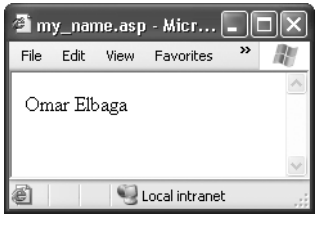


Figure 3-8. View of my_name.asp in the web browser after the ASP is code is processed

With the preceding code, you've concatenated the values to be displayed as one by using the & operator. You can also recreate a single variable to represent two variables concatenated. Modify the last example as follows, and you should get the same result as before:

```
<%
  myFName = "Omar"
  myLName = "Elbaga"

  myFullName = myFName & " " & myLName
  Response.Write myFullName
%>
```

The difference is in the code itself. By creating a new variable, you've made it simpler for yourself to display the full name—you don't have to concatenate both variables each time you want to display a user's full name.

You can concatenate variable and literal values too. For example, the following will also give you the same result as before:

```
<%
  myFName = "Omar"
  Response.Write myFName & " Elbaga"
%>
```

Watching out for "adding" numbers

Keep in mind that concatenation only joins things together; it doesn't add them mathematically in the case of numbers. For example, create a new ASP page named `adding_numbers.asp` and add the following code between the `<body></body>` tags:

```
<%
  myAge = 26
  yourAge = 31

  Response.Write myAge & yourAge
%>
```

When you view this in your browser, you'll see 2631, not 57 (the result of 26 + 31). If you want to add numbers together, you need to use a mathematical operator called the **Addition operator** (+), which adds numbers together. Now replace the ASP code in the `adding_numbers.asp` page with the following code and view the page in your browser:

```
<%
  myAge = 26
  yourAge = 31

  Response.Write myAge + yourAge
%>
```

This will give you an output of 57, as shown in Figure 3-9.

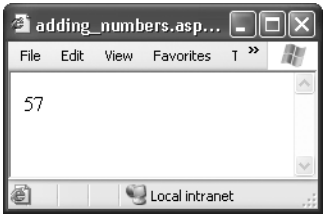


Figure 3-9. View of adding_numbers.asp in the web browser after the addition code is processed

Also be aware that although strings can contain numeric values, you won't be able to perform any mathematical computations on them—since they're enclosed in quotation marks, they aren't treated as numbers that can be computed. For example, the following code will give you 2631 again, not 57.

```
<%
  myAge = "26"
  yourAge = "31"

  Response.Write myAge + yourAge
%>
```

The two strings have been concatenated, not added together.

Variable naming conventions

This section is about how to name your variables. You can actually name your variables whatever you like, as long as you don't break any of the syntax rules, such as using a number as the first character of your variable. The other rules are mentioned in the previous section, "Syntax rules for variables," if you need a refresher.

Prefixes

Although you can name your variables whatever you like, there are some conventions you should follow to make your code more readable and consistent. For example, if a variable is a string, you might add an `str` prefix, making it `str_myvariable` or even `strmyvariable`—but the key here is adding a prefix so that you know what data type the variable is.

Sometimes, coders will add initial prefixes according to their names or companies, so that their code stands out. For example, Macromedia will often add the prefix `MM` to their variables, making them `MM_RedirectPage` or `MM_ConnectionString`, for example. I might add `OE` (my initials) to my variables. Prefixes make your variables more understandable, and can add a personal touch.

Using understandable variable names

Believe it or not, this is one of the most important aspects of variables, and for your coding in general! How you name your variables is key to the readability of your code. If you want to create a variable that will store a password, don't call it `p`. You may remember what it means for the next five minutes, but when you (or other developers) come back to update the code in six months, there's no way you'll

remember what p stands for. Use pwd or user_password instead. By the way, password is a reserved keyword in some languages, such as SQL.

The only time you might consider using a variable name like p is if you want to deliberately hide the variable name for security reasons—for example, if the variable name becomes exposed to a user of your site through a querystring.

You can learn more about querystrings at the Microsoft Windows 2000 Server Documentation site, found here:

www.microsoft.com/windows2000/en/server/iis/default.asp?url=/windows2000/en/server/iis/htm/asp/vbob53hj.htm

> *The best advice to give for naming variables is keep it simple, appropriate, and obvious. Avoid naming your variables solely by initials, such as* RP, ag1, *or* str_cdp, *unless there's a good reason to, which is unlikely. Make sure that what the variable represents is obvious, just from glancing at it, such as* RedirectPage, ArabicGreeting, *etc. After following these guidelines, feel free to add your own prefixes. For example,* str_RedirectPage *or* OE_ArabicGreeting.

Commenting code

Commenting your code allows you to display useful textual information about your code—to yourself or whoever has to sift through it—without it actually being parsed by the server; the server will just ignore the text or code that's commented out. Here's an example of an HTML comment:

```
<!-- This table is to hold the main navigation of the page -->
```

Everything within the <!-- and --> delimiters is ignored by the web browser. HTML comments are, however, viewable by users via the View ➤ Source option of their browsers.

ASP comments are created by putting a single quotation mark at the beginning of the line. This causes the line to be ignored by the server, even if it has actual ASP code in it. If you were to input the following ASP code within the body of an ASP page and run it in your browser, it would not appear in the source code for the web page:

```
<%
str_Username = "Omar"

'str_Username value holds the usernames of my web users.
' <%=str_Username%> is a shortcut to print out variables.
' Notice how the above line contains actual ASP code but
' doesn't execute.
' copyright ©, Omar Elbaga 2005

%>
```

ASP comments aren't viewable in the source code of the ASP page fed to the browser because they're never parsed and sent to the client browser; they're only viewable when the ASP page itself is opened on the actual server it resides on.

Dreamweaver displays all comments in gray in its Code view.

Commenting code is very important. It allows you to block out code quickly when you want to test the effect of a certain block of code. This is useful if you're trying to work out where an error lies. It also gives you the opportunity to document your code for yourself and others. Like using understandable variable names, this is helpful when you or others want to update your code.

> *Commenting your code allows you to provide useful information about your code, and it's important to do it as you're coding, while everything is fresh in your mind.*

VBScript's house of built-in functions

VBScript has a number of built-in functions that are at your disposal. They simplify things for you tremendously and give you the ability to manipulate variables as desired. There are different categories for built-in functions, as follows:

- **Conversion functions**: used to convert a value from one data type to another (such as from a string to an integer).
- **String functions**: perform actions on variables of the string data type.
- **Date/time functions**: deal with the display of the date and/or time.
- **Format functions**: used for formatting data according to how you want it to be displayed.
- **Math functions**: perform mathematical operations.
- **Array functions**: used to manipulate arrays. An array is like a list in which each item is referenced by a name and an index number (its location in the list).

There are around 100 functions—in this section, you'll be given a general overview of the most important ones. For a complete list of VBScript functions, check out the following websites: `www.w3schools.com/vbscript/vbscript_ref_functions.asp` and `http://devguru.com/technologies/vbscript/13896.asp`.

Conversion functions

This type of function converts a variable from one data type to another. For example, if you want to convert a number that's a string to an integer, you use the `CInt()` function. The variable is placed between the empty parentheses.

Create a new ASP page and save it as convert_to_int.asp. Insert the following code between the <body></body> tags:

```
<%
str_myAge = "26"
int_myAge = CInt(str_myAge)
'convert alphanumeric 26 to integer 26.

Response.Write int_myAge
%>
```

Now, when you print to the screen, you'll still see the result 26. That's because you made no changes to the value, but you changed the data type. The difference is that the variable can now be manipulated in a way that the string can't—that is, mathematical functions can now be performed on it. You can play with it to see for yourself.

If you don't believe me, let's use another function, TypeName(), that will display the data type of the new variable, so you're sure for yourself!

```
<%
str_myAge = "26"
int_myAge = CInt(str_myAge)
'convert alphanumeric 26 to integer 26.

Response.Write TypeName(int_myAge)
'see data type after conversion.
%>
```

If you enter the preceding code and print it to the screen, you'll see the following:

```
Integer
```

String functions

These functions allow you to do some cool things with strings. Let's first look at the string functions Len() and Trim().

The Len() function returns the number of characters in a string. Add the following code to the body of a new ASP page and save it as len.asp:

```
<%
myString = "I am cool!"
Response.Write myString
Response.Write "<br />There are " & Len(myString) &
➥ " characters in your string variable."
%>
```

View the page in your browser, and you should see the output shown in Figure 3-10.

len.asp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
I am cool!
There are 10 characters in your string variable.
Done
Local intranet

Figure 3-10. View of len.asp in the web browser

Keep in mind that even empty spaces count as characters—because they are!

The `Trim()` function gets rid of empty characters that may appear on the left and right sides of a string. This function is often useful when you're allowing web users to send text through a form (such as an e-mail submission form), during which additional empty spaces can sometimes get accidentally added to the text. When this happens, these spaces can cause problems in your database. The `Trim()` function is used to prevent this from happening.

Add the following code to the body of a new ASP page and save it as `trim.asp`:

```
<%
myString = " I am cool  "

Response.Write "." & myString & "."
Response.Write "<br />." & Trim(myString) & "."
'periods are concatenated with string so you can see the difference.
%>
```

View the page in your browser, and you should see the results shown in Figure 3-11.

Figure 3-11. View of trim.asp in the web browser. Notice the empty spaces between the periods and the text in the first line.

There are also functions to get rid of empty characters on the left or right alone: `LTrim()` and `RTrim()`. They're used in the same way.

Another function, `Left()`, allows you to return a specified number of characters from the left side of a string. It accepts the number as an attribute after the string. For example, `Left(string, 5)`.

For an example of the Left() function in action, add the following code to the body of a new ASP page and save it as Left.asp:

```
<%
myString = "I am cool. Didn't you know that I was valedictorian of my
High School class. Not only that, but I also have a
➥ Bachelor's Degree in Creative Writing."

Response.Write Left(myString, 10)
%>
```

You should see the following results:

```
I am cool.
```

Notice how only the first ten characters are returned. Many news websites do this kind of thing to display, say, only the first paragraph of several news articles on a summary page.

You should now have a general idea about VBScript functions and how to use them. As shown in the preceding examples, a variable is placed inside the parentheses of a function, and in many instances, additional attributes are also required (as in the case of the Left() function).

I would suggest keeping a VBScript reference book handy, as there is a lot to learn, and it's difficult to remember everything. The following websites are both valuable online references, each of which list the VBScript functions and include examples of their use:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vtorifunctions.asp

http://devguru.com/technologies/vbscript/13896.asp

Operators

Operators allow you to manipulate data stored in variables within your code. Operators have four main classes:

- **Assignment operators** store the value to the right of the operator inside the variable to its left.
- **Logical operators** join the expression on the left of the operator to the expression on the right in a conditional statement (you'll learn about conditional statements in the "Conditional logic" section later in the chapter)
- **Comparison operators** compare two arguments and check whether a specified condition is met.
- **Mathematical operators** perform a mathematical operation between the values on the left and right of the operator.

Assignment operators

There's only one assignment operator, and that's the equal sign (=). It simply stores the value to the right of the operator inside the variable to the left. You've been doing this throughout the examples in this chapter, so you should be used to it by now. Here's an example:

```
<%
Dim myGreeting
myGreeting = "Hola"
%>
```

The preceding assignment operator simply stores the string `Hola` inside the variable `myGreeting`.

Logical operators

There are three logical operators. Logical operators simply join together or manipulate Boolean (true or false) expressions in conditional statements.

Operator	Meaning
AND	Logical combination
OR	Logical separation
NOT	Logical negation

Let's have a look at each of these operators and see how they join and manipulate expressions.

AND, OR

The `AND` operator combines two expressions, making a result true only when both expressions are true. The `OR` operator separates the two expressions, making a result evaluate to true if either expression is true.

Let's see these in action. Create a new page and save it as `andor.asp`. Insert the following code within the body:

```
<%
number1 = 4-1
number2 = 6-2
If number1 AND number2 = 3 Then
Response.Write "Correct"
Else
Response.Write "False"
End If
%>
```

The preceding example checks whether both the variables `number1` and `number2` equal `3`. Since they do not both equal `3`, `False` is printed out, as shown in Figure 3-12.

Figure 3-12. View of andor.asp in the web browser. The result is False when the AND operator is used.

Now if you use the same code and change the `AND` to `OR`, what do you think the results will be? Try it and see. The result is shown in Figure 3-13.

```
<%
number1 = 4-1
number2 = 6-2
If number1 OR number2 = 3 Then
Response.Write "Correct"
Else
Response.Write "False"
End If
%>
```

Figure 3-13. View of andor.asp in the web browser when the OR operator is used

In the preceding example, when `AND` is replaced with `OR`, the code checks whether either variable `number1` or `number2` equals `3` (notice that `number1` equals 3 but `number2` does not). If either of them do, the word Correct prints to the screen.

NOT

The `NOT` operator simply negates the statement to the right of the operator. If an expression is `True`, `NOT` will change it to `False`, and if the expression is `False`, `NOT` will make it `True`. The following two code blocks give a good indication of how the `NOT` operator works. Create a new ASP page and save it as `NOT_1.asp`. Insert the following code between the `<body></body>` tags:

```
<%
number1 = 4-1
If number1 = 3 Then
Response.Write "Correct"
```

```
Else
Response.Write "False"
End If
%>
```

Since 4 – 1 = 3, `Correct` will print out. Now add the `NOT` operator, as shown in the following code, and see what happens:

```
<%
number1 = 4-1
If NOT(number1 = 3) Then
Response.Write "Correct"
Else
Response.Write "False"
End If
%>
```

This time, `False` is returned—`number1` still equals 3, of course, but the `NOT` operator changes the code to become the opposite of what it would otherwise be.

Comparison operators

Comparison operators compare two arguments and check whether a specified condition is met. The following table summarizes each comparison operator and its meaning.

Operator	Meaning
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

For example, consider the following code:

```
<%
x = 1
y = 3
If x < y Then
'Execute this code
End If
%>
```

This code checks whether the value of the variable x is less than the value of the variable y. If so, any code placed between the lines `If x , y Then` and `End If` will execute until the `End If` statement is reached.

Mathematical operators

Mathematical operators perform a mathematical operation between the data on the left and the right of the operator. The following table lists each mathematical operator and its purpose.

Operator	Meaning
^	Exponentiate
*	Multiply
/	Divide
\	Integer divide
MOD	Modulus
+	Plus
-	Minus
&	Concatenate

Here's a simple example of a mathematical function using the plus (+) operator:

```
<%
x = 54
y = 67
z = x + y
Response.Write z
%>
```

In the preceding code, the value of the variable z printed out. The value of z is the equivalent of the sum of x and y.

While the divide operator (/) divides the left-hand number by the right-hand number, returning the result plus the remainder, the integer divide operator (\) returns only the quotient, and modulus (MOD) returns only the remainder.

For example, 5 divided by 3 is 1 with a remainder of 2. The integer divide operator will return 1, while the modulus operator will return 2.

For an example of the divide, integer divide, and modulus operators, create a new ASP page and save it as `divide_operators.asp`. Insert the following code in the body of the page:

```
<%
number1 = 5/3 'Divide
number2 = 5\3 'Integer Divide
number3 = 5 MOD 3 ' Modulus
Response.Write number1 & "<br />" & number2 & "<br />" & number3
%>
```

View the page in your browser, and you should see the results shown in Figure 3-14.

```
1.66666666666667
1
2
```

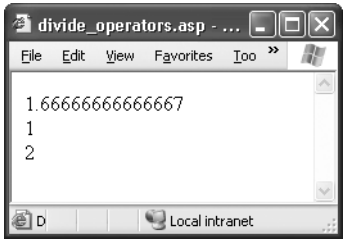


Figure 3-14. View of divide_operators.asp in the web browser

Operators are used all the time. You'll use them so much that they'll become second nature to you. However, it's a good idea to always have a reference manual available with you. For further reading on VBScript operators, visit `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vbsoperators.asp`.

Operator precedence

As you know, ASP code executes from left to right, line by line. But there's an exception to this—some operators execute according to their own order, known as **operator precedence**. While comparison operators all have equal precedence (and thus execute from left to right), mathematical and logical operators execute in their own order of precedence, regardless of their position in a line. The following table lists the order of precedence of mathematical and logical operators.

Operator type	Operation	Precedence order
Mathematical		
	Exponentiate (^)	1
	Multiply (*), divide (/), integer divide (\)	2
	Modulus (MOD)	3
	Add (+), subtract (-)	4

Operator type	Operation	Precedence order
Logical		
	NOT	1
	AND	2
	OR	3

Note that operators listed on the same line of the preceding table are of equal precedence; they will be processed in left-to-right order when they appear in the same line.

As an example, create a new ASP page and save it as `operator_order.asp`. Insert the following code in the page and view it in your web browser:

```
<%
  intOrderTest = 3 + 2 * 4
  Response.Write intOrderTest
%>
```

In left-to-right order, the answer you would expect is 20, since 3 + 2 = 5, and 5 × 4 = 20. However, since multiplication has a higher precedence than addition, the answer will be 11. With the multiplication command executing before the addition, the equation is actually 2 × 4 + 3. Using parentheses to group expressions allows you to force operators to execute before those with a higher precedence. For example, to force addition to occur before multiplication in the preceding expression, you would replace the code in `operator_order.asp` with the following:

```
<%
  intOrderTest = (3 + 2) * 4
   Response.Write intOrderTest
%>
```

Figure 3-15 shows the new result.

Figure 3-15. The new result from operator_order.asp, this time using parentheses to override operator precedence

Expressions in parentheses are always evaluated before expressions that aren't in parentheses, regardless of operator precedence. You should always group mathematical expressions in parentheses, even if the natural order is what you desire, simply to make your code more readable.

Conditional logic

Like variables, conditional logic is one of the most important aspects of the language, because it allows you to control the flow of your application according to certain criteria—in particular, according to how web users interact with your code. Conditional logic (also known as conditional statements) allows you to use code that says things like "If this happens, then I want you to do this, but if it doesn't, do that instead." Let's take a look at how you can control the interaction you receive with conditional logic.

The If statement

The `If` statement is the mother of all control. It says, "If this is true, then do this; otherwise, do that." Take a look at the following example. Create a new page and save it as `ifstatement.asp`. Enter the following code anywhere between the `<body></body>` tags:

```
<%
myExpectedOutput = "Cool"
If myExpectedOutput = "Cool" Then
Response.Write "That's what I thought!"
End If
%>
```

Figure 3-16. View of ifstatement.asp in the web browser

View the page in your web browser, and you should see the results shown in Figure 3-16.

In the preceding code snippet, the `If` statement checks to see whether the `myExpectedOutput` variable equals your desired value; and when it does, you can specify accordingly what action you want to take place when you receive an expected output. In the previous example, the action taking place is the printing of a related message to the user. Keep in mind that when comparing two strings, every character in each string (including spaces) must match for them to be considered equal.

You can also set the conditions around embedded HTML by using multiple code blocks, as follows:

```
<%
myExpectedOutput = "Cool"
If myExpectedOutput = "Cool" Then
%>

That's what I thought!

<%
End If
%>
```

This will return the same results as the previous code. However, if the condition is not true, the HTML won't display. This is often used when you want to check if a user is logged in. To do this, you could use an `If` statement to verify whether a cookie you previously set exists. If it does, you can allow a user access to a page; if it isn't, you can simply redirect the user to another page to prevent access.

The If . . . Then . . . Else statement

You can also add the Else keyword to the statement to introduce another output if the expression doesn't meet the expected criterion. Create a new ASP page and save it as expectedoutput.asp. Enter the following code between the <body></body> tags:

```
<%
  myExpectedOutput = "This web site sucks!"
  If MyExpectedOutput = "Cool" Then
  Response.Write "That's what I thought"
  Else
  Response.Write "Hey, think again."
  End If
%>
```

This will output to the web browser the results shown in Figure 3-17.

In the preceding code, the If statement checks to see whether the myExpectedOutput variable equals your desired value. If it doesn't, the Else statement causes the comment Hey, think again. to be output. Again, you can set these conditions around embedded HTML using multiple code blocks, as in the previous example.

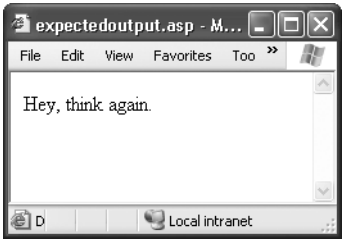


Figure 3-17. View of expectedoutput.asp in the web browser

The If . . . Then . . . ElseIf statement

The If...Then...ElseIf statement allows you to provide two or more possible code continuations. This statement is preferable to using multiple If...Then...Else statements, which can result in inefficient and sloppy code. The ElseIf statement is beneficial in that it allows you to combine multiple If...Then...Else statements in one. Take a look at the following example. Create a new ASP page, save it as user_rating.asp, and enter the following code between the <body></body> tags:

```
<%
userRating = 9

If userRating = 4 Then
Response.Write "Are you serious?"

ElseIf UserRating = 6 Then
Response.Write "C'mon... you can do better than that."

ElseIf userRating = 9 Then
Response.Write "Close, but I deserve better."

ElseIf userRating = 10 Then
Response.Write "There you go!"

End If
%>
```

Since the value of the userRating variable is set to 9, this code will output the results shown in Figure 3-18.

Figure 3-18. View of user_rating.asp in the web browser

The preceding code can be used to allow a web user to enter a rating into a form, and then be returned the appropriate response. Let's see it in action.

1. Create a new dynamic ASP VBScript page and name it user_rating.asp. Feel free to overwrite the last ASP page you created.
2. Switch to Design view and create a simple form with one text field and one Submit button.
3. Name the text field userRating and set the action to go back to the same page.
4. Now switch to Code view and insert the following code anywhere between the HTML <body></body> tags:

```
<%
userRating = Request.Form("userRating")

If userRating = 4 Then
Response.Write "Are you serious?"

ElseIf UserRating = 6 Then
Response.Write "C'mon... you can do better than that."

ElseIf userRating = 9 Then
Response.Write "Close, but I deserve better."

ElseIf userRating = 10 Then
Response.Write "There you go!"

End If
%>
```

5. Now load the page in your web browser. Try entering 4, 6, 9, and 10, and see how the responses change with each different entry (see Figure 3-19).

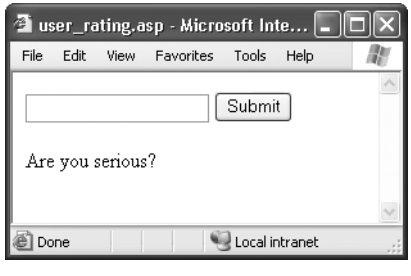


Figure 3-19. View of user_rating.asp in the web browser when a 4 is submitted through the form

You can use an Else keyword for your last check in an ElseIf statement. It gets processed only when none of the other conditions match. Be aware that if you use it, you must put it at the end of the statement, and never before an ElseIf keyword. The following example shows the correct use of the Else statement in such a case:

```
<%
userRating = Request.Form("userRating")

If userRating = 4 Then
Response.Write "Are you serious?"

ElseIf UserRating = 6 Then
Response.Write "C'mon... you can do better than that."

ElseIf userRating = 9 Then
Response.Write "Close, but I deserve better."

Else
Response.Write "I'm looking for a 10 rating."

End If
%>
```

Looping logic

Technically, a loop statement allows you to loop through an action while or until a condition you specify is met. In other words, it gives you the ability to have an action occur over and over until you want it to stop. For example, you may want to loop through the code and list a set of variables or a set of records listed in a database table. In VBScript, there are several different types of loop statements that for the most part accomplish the same task. I'll demonstrate two of the most commonly used ones.

The Do . . . Loop statement

This loop allows you to create a loop while a condition you specify is true. If the statement could talk, it would say something like this:

```
Do While your condition
Code to be executed here
Loop
```

What will happen is your code will keep going according to the condition you specified. It will stop when the condition is met or not met, depending on the type of condition you specify. You might tell it to keep printing a database record to the screen while the record doesn't equal a particular value. You can also tell it to keep printing a database record until it equals, say, 10.

Let's take a look at a real example. Create a new ASP page, save it as dowhile.asp, and insert the following code between the <body></body> tags:

```
<%
Dim intCount
intCount = 1

Do While intCount <= 15
Response.Write intCount
intCount = intCount + 1
Loop
%>
```

View the page in your web browser, and you should see the results displayed in Figure 3-20.

The preceding example basically tells the server to start the variable intCount off with a value of 1, print it out, add another 1 to it, and keep doing this until the variable equals 15.

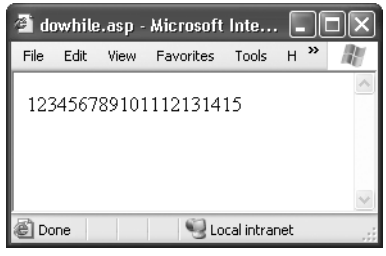


Figure 3-20. View of dowhile.asp in the web browser

The While . . . Wend statement

Dreamweaver uses this loop in the Repeat Region behavior for looping through database records. This loop works exactly like the Do...Loop statement. To achieve the same results as the previous example, you simply replace Do While with While, and Loop with Wend, as shown in the following code:

```
<%
Dim intCount
intCount = 1

While intCount <= 15
Response.Write intCount
intCount = intCount + 1
Wend
%>
```

For further reading about VBScript loop statements, see the following page on the MSDN website: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vbsLooping.asp

Cookies (ASP cookies, not chocolate chip!)

Whether you're an experienced web developer or simply an Internet enthusiast, you've no doubt heard about cookies. Cookies aren't our favorite snack food in this context—rather, they're simply a term used to describe another great feature, which is not only native to ASP, but usable with almost any web scripting language. (Netscape created the original cookie.)

Cookies are simple text files that you can create and store on the client machine. They allow you to store simple data such as usernames, dates, etc. Values of cookies can be created by the web application, or they can accept parameters submitted by the user. However, they must be used with care.

How do you actually make use of them? Cookies are comparable to the forms and URL parameters discussed earlier—they allow you to maintain data across pages. When cookies are created, they're automatically stored in a specific folder in the user's hard drive, which can't be changed—in Windows 2000, you can find this folder in `C:/Documents and Settings/Administrator/Cookies`. You'll probably find dozens already there. Each cookie contains specific data that was stored by various web applications while you were using the Net.

Response.Cookies and Request.Cookies

The ASP statement `Response.Cookies` is all you need to create a cookie. You just need to name the cookie and give it a value—for example, `Response.Cookies("MyCookie") = "Hello World"`.

In this example, the cookie is called `MyCookie` and it has a value of `Hello World`. Once the code executes, the cookie will be created. The preceding example uses a specific value, but remember that you can also store information from the user as a value. For example, if a user submits a form with a text field called username, you can store the value of that text field as the value of a cookie, like so:

```
Response.Cookies("MyCookie") = Request.Form("username")
```

This way, you can store the form data on the user's computer and retrieve it any time (as long as it hasn't been deleted).

At this point, you're probably asking, "Now that I know how to create a cookie, how do I retrieve it?" `Request.Cookies` will retrieve any cookie you created, identifying it by name. For example, if you wanted to retrieve the cookie named `MyCookie` that was mentioned earlier, you would do it like so: `Request.Cookies("MyCookie")`. How easy is that? Then, if you want to output it onscreen, you use the following code: `Response.Write Request.Cookies("MyCookie")`.

Let's have a look at an example to put this theory into practice. Create a new ASP VBScript page and save it as `my_first_cookie.asp`. Enter the following code between the `<body></body>` tags:

```
<%
Response.Cookies("username") = "omar"
%>

<%=Request.Cookies("username") %>
```

You should see the following results when you load the page in your browser:

omar

You created a cookie named username and gave it a value of `omar`—then you displayed the cookie.

Now let's create some cookies based on what a user might submit in a form. You'll use another dummy login form for this example. Create a new page and save it as `login_create_cookies.asp`. Enter the following code between the `<body></body>` tags:

```
<form name="form1" method="post" action="login_create_cookies.asp">

username: <input type="text" name="username"><br>
password: <input type="text" name="password"><br>
<input type="submit" name="submit" value="submit">

</form>

<!-- login_create_cookies.asp -->

<%
Response.Cookies("userid") = Request.Form("username")
Response.Cookies("pwd") = Request.Form("password")
%>
userid cookie: <b><%=Request.Cookies("userid") %></b><br>
pwd cookie: <b><%=Request.Cookies("pwd") %></b>
```

After submitting the data, when you view the `login_create_cookies.asp` page, you should see the values you entered. You've seen something similar before, but this time you didn't display the values of the text entered in the text fields by simply using `Request.Form`—rather, you stuffed those values into two cookies and then retrieved the values from each (see Figure 3-21).

Figure 3-21. View of login_create_cookies.asp in the web browser after the form is submitted with sample username and password

Now that these cookies have been created, you can retrieve the data from any page without having to prompt the user again (until the cookies are deleted). This comes in handy if you wanted to, say, display the username across several pages.

> *Many web developers use a prefix of* `ck` *when naming cookies. For example,* `ckUserID`*,* `ckPassword`*, etc.*

Cookie expiration

If you don't specify a cookie's expiration date, it will delete itself from the user's system once the user shuts down his or her browser. This is fine if you only want the cookie information for a single session, but you may want the information to be maintained for days, weeks, or even years.

For example, let's say you want a cookie named `userid` to expire after seven days. You would do this with the following code:

```
Response.Cookies("userid").Expires = Date + 7
```

Likewise, you can add expiration dates to your first cookie example in the `my_first_cookie.asp` page like so:

```
<%
Response.Cookies("username") = "omar"
Response.Cookies("username").Expires = Date + 7
%>

<%=Request.Cookies("username") %>
```

You can increase the number of days as you require.

> *Keep in mind that even when you set expiration dates for your cookies, users are still likely to manually delete them from their computers every so often.*

Updating cookies

You can't create two cookies on the same computer with the same name, but you can update cookies simply by rewriting their values. The following example will show you how to do this.

Create a new ASP VBScript page named `rewrite_cookie.asp`. Enter the following code:

```
<%
Response.Cookies("ckRememberMe") = True
Response.Cookies("ckRememberMe") = False
%>

<%=Request.Cookies("ckRememberMe") %>
```

If you view the page, you should see the following results:

```
False
```

In the preceding code, you created a cookie called `ckRememberMe`, gave it a value of `True`, and then rewrote the value to `False`.

Deleting cookies

You can delete cookies simply by giving them an empty string value. Here's an example that shows you how to do this.

Create a new ASP VBScript page named `delete_cookie.asp`, and enter the following code:

```
<!-- delete_cookie.asp -->

<%
Response.Cookies("ckRememberMe") = True
Response.Cookies("ckRememberMe") = ""
%>

<%=Request.Cookies("ckRememberMe") %>
```

Nothing should display on the screen, since the cookie doesn't exist. ASP is cool in that it doesn't throw an error when trying to retrieve a cookie that doesn't exist.

Cookies are cool and fun. They have many uses and can help web developers in many various situations—don't be afraid to use cookies if you see a good use for them in your web applications. You may also want to explain to your users how you use cookies to track their interaction on your site in a privacy notice or disclaimer.

A word of warning about the use of cookies

When using cookies, you should be careful not to use them to store any sensitive information, such as usernames and passwords, or—heaven forbid—credit card details! Cookies aren't dangerous *per se*—but it's the potential of *how* they can be used that makes them a security risk. Since they're merely unencrypted text stored in a file on a user's computer, it's a possibility that some unscrupulous web users could access them (as long as they have the cookies' names), and use them for untoward purposes, which might include sharing the cookie information with companies that users don't want to be contacted by.

However, as long as you're wise in your use of cookies, and heed these warnings, you should be fine—besides, cookies are beneficial to any web developer. They enable persistent logins and personalization. They really give you an edge when you know when and how to use them. What's more, they don't take up server space like sessions, which are stored on the server.

> *For more specific information on cookies and their potential risks, visit Netscape's legal notice on cookies at* `www.netscape.com/legal_notices/cookies.html`.

Session variables

Session variables provide another way to maintain state across several pages within your web application. When a user accesses a page on your site, memory is allocated on the web server for this user

inside what's called a **session object**. Any variable stored in this session object is called a **session variable**. Session variables can be used to maintain state for the duration of a user's visit to the website. You can store values in session variables that are **unique** to each user, just the way cookies are. Every user gets his or her own session when they access your website.

Unlike cookies, sessions live on the server, not on the client's computer. They die if there's no request from the user to the server within a session timeout interval, set either in IIS (Internet Information Services) or an ASP page.

Setting the session timeout interval

In IIS, 20 minutes is the default timeout for sessions, which you as the web developer can change if you like. To do so, go to the Windows Control Panel and open Internet Information Services from the Administrative Tools. Expand the tree until you can see your default website. Right-click Default Web Site. Choose Properties from the shortcut menu. Select the Home Directory tab, click the Configuration button, and select the App Options tab. Make sure the Enable session state option is checked. Set the session timeout in minutes. Click OK to exit the Application configuration dialog, and OK again to exit the Properties dialog.

You can also programmatically set the session timeout from within an ASP page by adding the following code at the top of your ASP page:

```
<% Session.Timeout = 10 %>
```

Creating and retrieving session variables

Session variables are created just as easily as the other variables. You can write a value into a session variable with the following code:

```
Session("session_name") = "my value"
```

Retrieving session variables is also very easy. You simply need to print out the session—for example, like so:

```
Response.Write Session("session_name")
```

Let's create an example similar to the previous cookie example, but instead of storing the values in cookies, you'll store them in session variables. Create a new ASP VBScript page, save it as `login_create_sessions.asp`, and insert the following code:

```
<form name="form1" method="post" action="login_create_sessions.asp">

username: <input type="text" name="username"><br>
password: <input type="text" name="password"><br>
<input type="submit" name="submit" value="submit">

</form>

<!-- Now create a new file and save it as
➥ login_create_sessions.asp and insert the following code.-->
```

```
<%
Session("userid") = Request.Form("username")
Session("pwd") = Request.Form("password")
%>
userid session: <b><%=Session("userid") %></b><br>
pwd session: <b><%=Session("pwd") %></b>
```

Next, access login_create_sessions.asp, enter some values, and submit the form. After submitting the data, when you view the login_create_sessions.asp page, you should see the values you entered (see Figure 3-22).

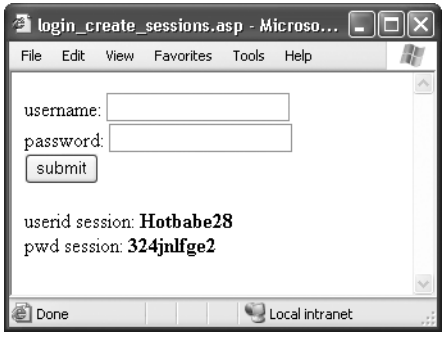


Figure 3-22. View of login_create_sessions.asp in the web browser after the form is submitted with a sample username and password

Updating session variables

Session variables can be updated simply by rewriting the value, just like with cookies. The following example shows how it's done.

Create a new page, save it as rewrite_session.asp, and insert the following code in the body of the page:

```
<%
Session("svRememberMe") = True
Session("svRememberMe") = False
%>

<%=Session("svRememberMe") %>
```

View the page, and you should see False.

Deleting session variables

You can destroy a particular session variable with Session.Contents.Remove. For example, the following code would destroy a session named MM_Username:

```
<% Session.Contents.Remove("MM_Username") %>
```

You can also destroy all values stored in the session object, which will destroy all sessions on the server that were set for the user, by using `Session.Abandon`. Use this with caution because it will delete *all* session variables associated with the user:

```
<% Session.Abandon %>
```

If you send the user to a page that contains this code, the user's sessions will be destroyed and memory will be freed. For example, if you create a page named `delete_sessions.asp` with the preceding code, and then access it and go back to your `rewrite_session.asp` page, you'll find the sessions empty.

You can also destroy the entire session object with the following code:

```
<% Session.Contents.RemoveAll %>
```

Note on sessions

Be mindful when using sessions. Because sessions can take up a lot of server memory, you should use them sparingly, especially if you're expecting a lot of users at your site. One problem that arises with session variables is that it's sometimes difficult to determine how long to set the timeout interval. For example, if the session holds login information and the timeout interval isn't long enough, users will often have to log in repeatedly due to their session timing out. On the other hand, leaving the interval too long will eat up server resources, even if the user is no longer at your site.

Sessions last according to their timeout scope. This means that if a user logs on to your site, therefore creating a session, and then leaves after 1 minute, that session will still last for 19 more minutes regardless, eating up your server memory. For this reason, the length of session timeouts should be kept relatively short. Anything over 25 minutes is way too much. The key is using a balanced timeout interval and destroying the session in your ASP page once you no longer need it. Don't let this scare you too much. As you become more experienced, you'll know when and how to use them more efficiently.

Environment variables

Environment variables are hidden pieces of information about the web server and the visitor, which the server makes available to any program that requests them. These variables can offer very useful information, such as the name of the web server or the user's IP address. You can then access these variables within your ASP script. The following table shows some of the more commonly used environment variables.

Environment variable	Description
`HTTP_REFERER`	The URL of the page that sent the user to the current one
`QUERY_STRING`	The URL parameters following the ? in a URL
`REMOTE_HOST`	The client's IP address

Continued

Environment variable	Description
SCRIPT_NAME	The URL of the current page
SERVER_NAME	The server hostname
URL	The URL of the ASP page (excluding the querystring)

For a list of the many more environment variables, visit the following website: www.aspfree.com/asp/servervariables.asp.

You can retrieve a specific environment variable using the following syntax:

```
Request.ServerVariables("VariableName")
```

Let's look at an example of environment variables in action. Create a new page, save it as link_page.asp, and insert the following code to utilize some of the environment variables:

```
<a href="http://localhost/env_variables.asp">link to env_variables.asp</a>
```

Now create a new page, save it as env_variables.asp, and insert the following code in the body of the page to view some of the environment variables (see Figure 3-23):

```
The URL of this page is: <strong>
➥ <%=Request.ServerVariables("URL") %></strong>
<br />
Your IP address is: <strong>
➥ <%=Request.ServerVariables("REMOTE_HOST") %>
</strong><br />
Go back to the page you came from: <strong>
<a href="<%=Request.ServerVariables("HTTP_REFERER")%>">
➥ <%=Request.ServerVariables("HTTP_REFERER") %></a></strong>
```

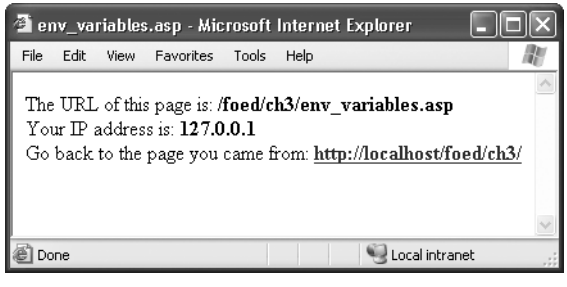

env_variables.asp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
The URL of this page is: /foed/ch3/env_variables.asp
Your IP address is: 127.0.0.1
Go back to the page you came from: http://localhost/foed/ch3/
Done
Local intranet

Figure 3-23. View of env_variables.asp in the web browser

Environment variables are beneficial in that they can give you pertinent information. For example, you could put the user's IP address as the value of a hidden field in a feedback form so that you have the IP address of the submitter. This is helpful in identifying a particular user. Once you have the user's IP address, you can use it to ban the user from your site or particular web application if necessary. It can also be reported to appropriate authorities on the account of abuse or security breaches. The following code shows such an example:

```
Send us a note!

<form name="form1" method="post" action="thank_you.asp">

  name: <input type="text" name="name"><br />
  email: <input type="text" name="email"><br />
  message: <br><textarea name="message" rows=2 cols=20
➥ wrap=hard></textarea><br />
  register for newsletter: <input type="checkbox"
➥ name="newsletter" value="1">
  <br />
  <input type="submit" name="submit" value="submit">

  <input type="hidden" name="ipaddress"
➥ value="<%=Request.ServerVariables("REMOTE_HOST") %>">
</form>
```

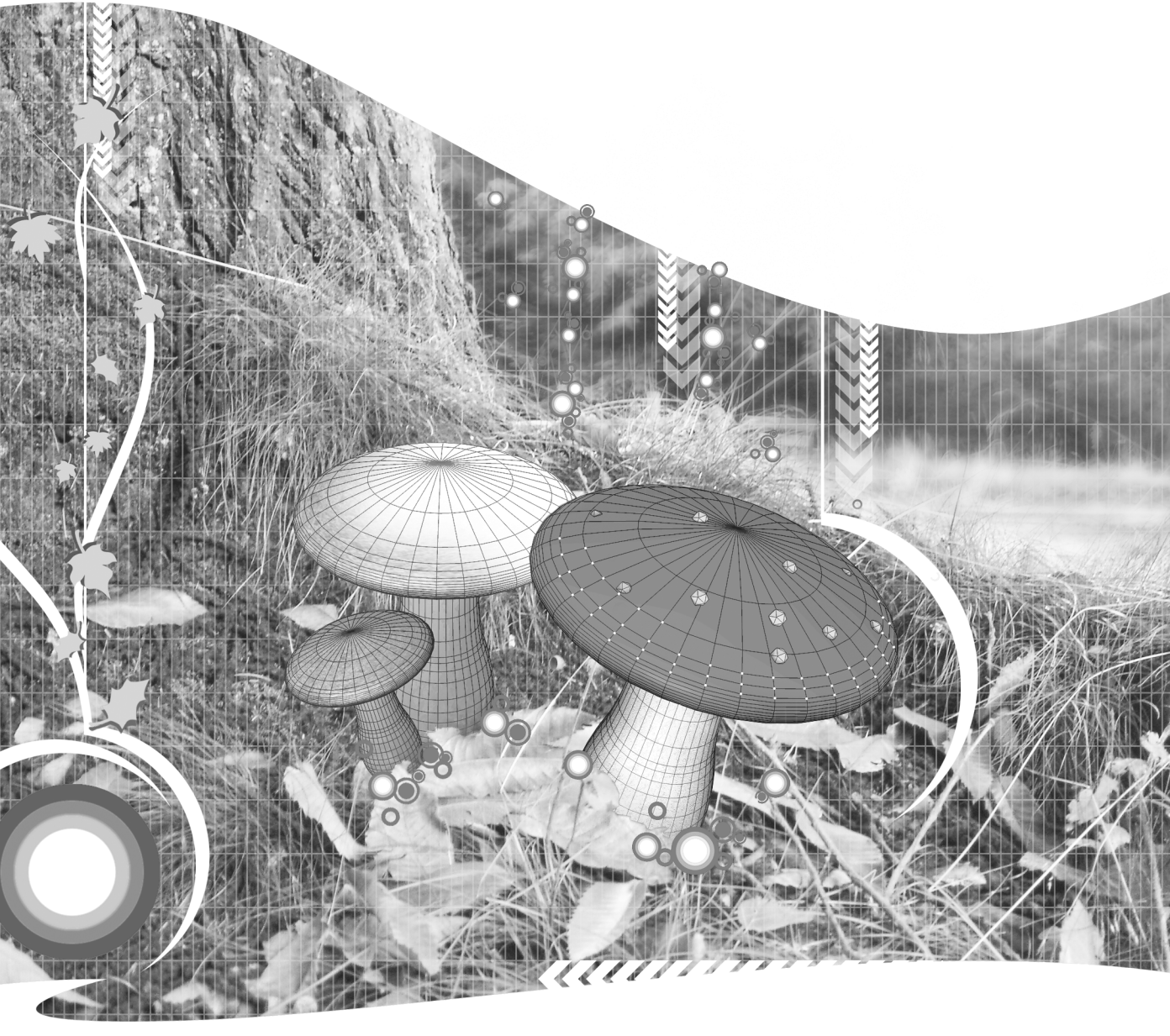
When you run this page, view the HTML source code to see the form's hidden element value so you can see that your IP address is in fact being stored in the form, ready to be submitted with the other fields.

Feel free to make use of these variables!

Conclusion

In this chapter, you learned the basics of the ASP VBScript language and how it's used to hand-code ASP pages. Although Dreamweaver will take care of much of the drudgery involved in hand-coding, it's important to know how to hand code some VBScript, especially when you want to start customizing some of your dynamic Dreamweaver ASP pages. You'll then need to delve into the code yourself and play around with it.

Learning the basics of VBScript is only the beginning. In the following chapter, you'll learn about one of the most important aspects of a dynamic web site: how to design a database and connect it to your ASP pages.



Chapter 4

DATABASES

People generally think that a database is something that's created in an expensive piece of software like Microsoft Access. However, a database could be defined as any collection of data that's organized in such a way as to make easy entry and retrieval possible. This means that files created in Notepad or Excel could be feasibly used as databases—although when used as such, we tend to refer to them as "datastores" to make it clear that they simply *store* data, leaving it up to us to organize how we go about updating or retrieving it.

Using a database application for storing your data has the advantage of providing a surrounding network of code for accessing the data—something a flat text file, for instance, does not. If you need ease of use from your database application (and let's face it, who doesn't?), then you should be looking to take advantage of a database application, such as the two you're going to look at in this chapter (and throughout the book): **Microsoft Access 2003** and **Microsoft SQL Server 2000**. This chapter will cover database-related activity with details relevant to both of these databases, as well as data interaction between Dreamweaver and the databases. Specific topics will include the following:

- Creating a database
- Making a SQL Server database Internet-ready
- Creating a SQL Server database user
- Understanding database design

- Creating tables
- Understanding relational databases and referential integrity
- Creating relationships
- Using views and queries
- Understanding fundamental SQL and using basic SQL commands
- Setting up DSNs to connect to an Access or SQL Server database
- Connecting to a database from Dreamweaver
- Building recordsets in Dreamweaver using the Simple and the Advanced Recordset builders
- Using the Dreamweaver Command dialog to insert, update, and delete records in a database

Creating a database

First, we need to establish the difference between creating a database and creating the database content. When you create your database, you're creating the container in which your data can reside. No data will be created at this point. Only after you've created the container can you proceed to creating the content inside it.

Inside Microsoft Access

Microsoft Access offers up myriad wizards and templates for you to choose from to get your job done. Creating a database is as simple as clicking the mouse a few times (and pressing a couple of keys too, unless you want to create a database called `db1.mdb`, which is Access's default name for the first database it's asked to create).

When you open Access, you're presented with a blank screen and a sidebar titled Getting Started. This sidebar contains a few useful elements, including a search facility to search the Help system both on and offline, a list of recently opened Access databases, and a link to create a new database (see Figure 4-1).

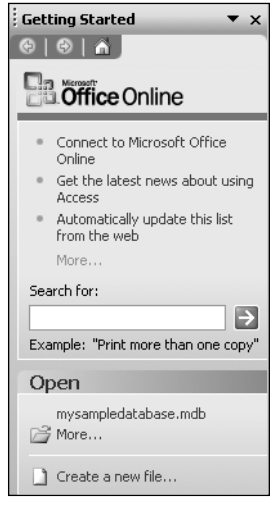


Figure 4-1. The Getting Started panel in Microsoft Access 2003

To create a new database, click the Create a new file link in the sidebar. Alternatively, you can select File ➤ New, or click the New icon on the icon bar below the menu. The sidebar turns into the New File panel, where you can now make your next choice (see Figure 4-2).

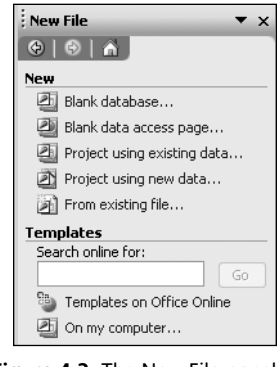


Figure 4-2. The New File panel

The New File panel offers you a few options to choose from, but you're only concerned with creating a new blank database at this point, so click the first option, Blank database. Then select a location to save the database on your computer, such as a databases folder in the root of the `C:` drive, `C:\databases\`, and give it a name (see Figure 4-3). The database name used for this book is `mysampledatabase.mdb`—just so you know it's a sample!

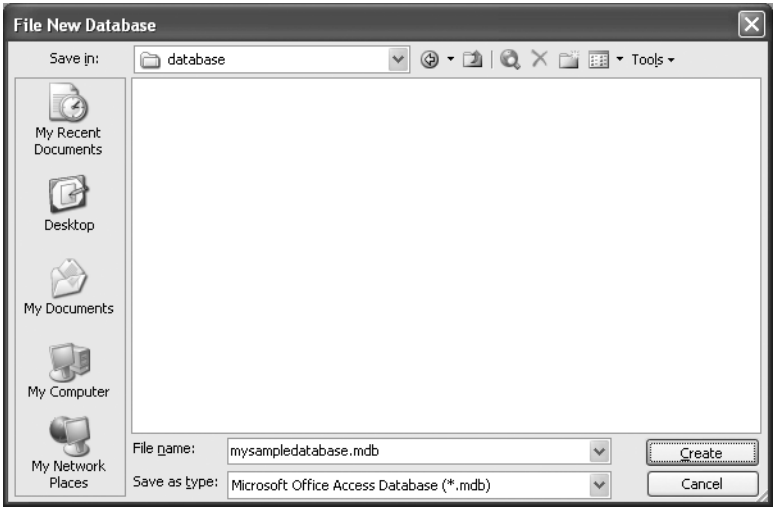


Figure 4-3. Specifying where to save your new file

Your new blank database will now open, ready for you to begin adding structure and content to it (see Figure 4-4).

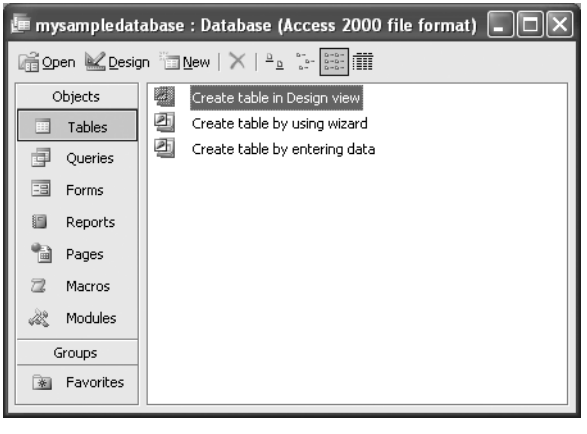


Figure 4-4. The new blank database

Before moving on to adding the content of your database, however, let's first cover the creation process in SQL Server.

Inside Microsoft SQL Server

You'll use **Enterprise Manager**, one of SQL Server's main interfaces, to create your database in SQL Server. If you prefer, you can also use the **Query Analyzer** to create your databases (using the SQL command `CREATE DATABASE`, for instance). In fact, SQL queries can be written into the Query Analyzer to perform everything else you could possibly think of with your database (and possibly lots

you might not think of too). However, although it's an extremely powerful tool, you're not going to use the Query Analyzer here because the benefits it offers really lie beyond your needs at this stage.

Begin by launching Enterprise Manager.

> *If you need to log on to your SQL Server, please ensure that you're logged in as Administrator, since this affects all the objects that you're going to create. By logging in as Administrator any object that you create will be prefixed with* dbo*., meaning that* dbo*—the database owner—is recorded as the owner of that object.*

Enterprise Manager runs under **Microsoft Management Console** (**MMC**). When the console opens, you'll see two panes. The left pane is a tree view, and the right pane shows the contents of the left pane's selected element (see Figure 4-5).

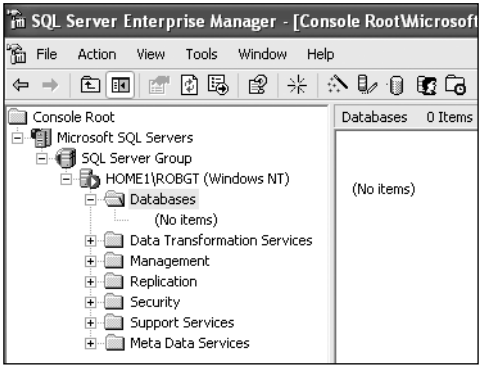


Figure 4-5. The tree view of objects in Enterprise Manager

Once you've expanded your tree view to look something like the one in Figure 4-5 (your server name will no doubt be different to mine), right-click the Databases node and select New Database from the context menu.

At a very basic level, all you need to do to create your SQL Server database is enter a name in the Name box on the General tab and click OK, which in many cases will be perfectly acceptable (see Figure 4-6).

A time that it may not be acceptable, however, is when your network administrator specifies the places that you must create all your files. In such a case, leaving the default locations of the Data files and the Transaction log files in place is not an option. To change the location of these files, you need to click each tab in turn and use the ellipses button (. . .) in the Location column to select the new storage location. You can also specify alternative names for the files here, instead of the default database names, which will have been automatically entered for you if you've already entered the database name on the first tab.

So, unless you have this worry, simply enter mysampledatabase as the name for your new database, leave all the defaults in place, and click OK.

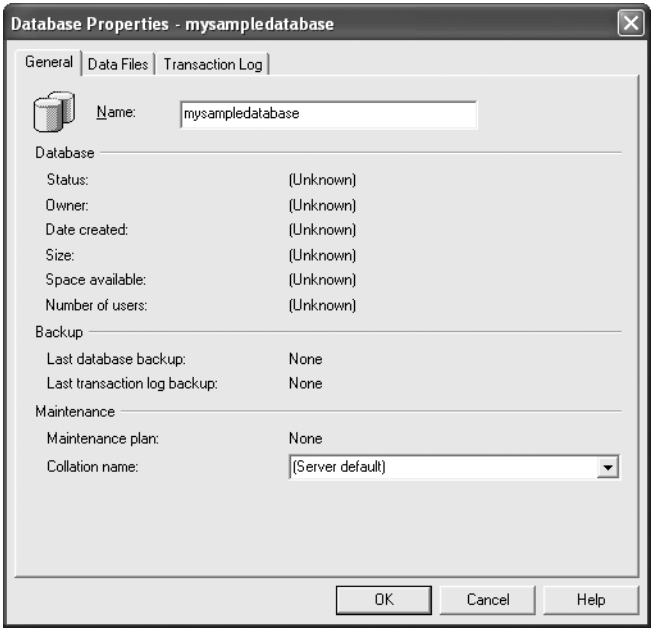


Figure 4-6. The Database Properties dialog box

A little explanation of what SQL Server is going to do with your data is in order at this point—stay with me.

One of the main differences between Access and SQL Server is the way data is managed. Access uses a single file (the .mdb file) to store everything—the limitations of which will become apparent shortly—whereas SQL Server uses \at least two files—a ***data*** *file and a* ***transaction log*** *file.*

Without getting too deep into the workings of SQL Server (which isn't within the remit of this book), what basically happens is the following: a user logs in to SQL Server and performs some actions on the data in one of its databases. Those actions are written to a transaction log and applied to the data at a subsequent point. If anything should go wrong while this user is interacting with the database, and his or her modifications can't be completed successfully, the transaction won't be applied, and the data will remain unaffected (this is called "transactional rollback").

SQL Server locks its data at the row level so that many users can use the same database—and even the same data—simultaneously. It can then schedule multiple user access requests to that data. It does this by order of merit (depending on what action is to be performed on the data) and by the order in which those requests were originally made. This is known as "concurrency." (You can read more about this subject in ***Books Online****—SQL Server's impressive help system, which can be found under the Microsoft SQL Server menu.*

Continued

Access, on the other hand, doesn't have this ability. Because it was designed for desktop use, Access locks its data at the database level, so that only one user can use the database and make modifications to the data at any one time. This may seem restricting (and it can be), but if your web traffic is low, this one-user-at-a-time principle shouldn't be noticeable. A web page uses a single database connection—regardless of how many people are interacting with the site—so as far as the database is concerned, it's the same user making data requests each time.

This is one of the reasons why Access is widely used as a database manager for small websites—despite its limitations, it holds up remarkably well under small loads. If you're just learning, or you expect very low web traffic on your website (for example, if it's a personal website), you might consider using it. That said, for anything resembling mainstream work in which performance and reliability are paramount, you would be well advised to use a tool like SQL Server, which was designed for such work.

Expanding the Databases node and your newly created database, which is now listed in the left pane of the SQL Server console, will show you its internal structure (see Figure 4-7).

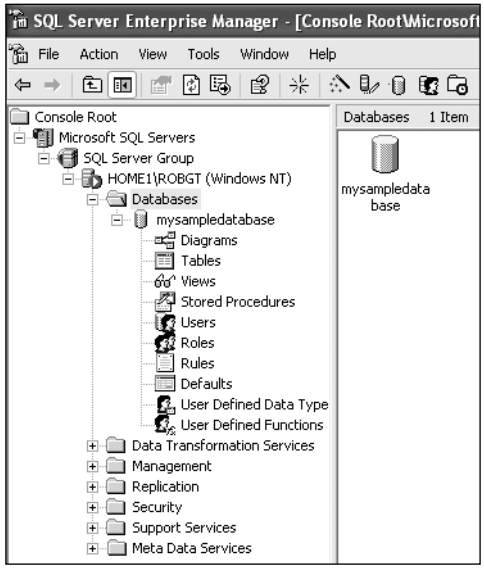


Figure 4-7. The new database and its internal structure of objects

Preparing a SQL Server database for Internet use

A new SQL Server database is created containing ten objects, two of which are prepopulated with data. If you click the mysampledatabase database name under Databases in the left pane, you'll see the items associated with each database listed on the right—you can change this view by right-clicking in the pane and selecting View.

The Users object contains an item called dbo (database owner). This object stores the database owner's account, and is created so that this user can administer the new database. For security reasons, other user accounts that are added to your database should only be given enough access for them to get their required job done. This prevents the risk of accidental damage being done by someone who has too many permissions. You can assign permissions on a new account manually, on an object-by-object basis, but it can be easier to assign a new account to one of the predefined **roles** that SQL Server makes available. A role is an array of permissions that are preset for each of the database objects, and represents one of a number of common database-related job roles. Roles become very useful if you're creating many user accounts.

> *To reduce clutter in the left pane, you may want to exclude from view all the system databases and objects. To do this, right-click the SQL Server name and select the* Edit SQL Server Registration properties *option. In the resulting dialog box, there are three check boxes at the bottom. You need to deselect the middle one marked* Show system databases and system objects. *Now you'll only see the items that you create yourself.*

You need to set up a special account in SQL Server in order for your web application to be able to talk to your database. This is the IUSR\<machine_name> account—the **Internet User** account. This account acts as an alias for all the traffic brought to the database from your web application.

Creating the IUSR account in SQL Server

1. In the left pane of the SQL Server console, click Security, and then click the Logins node.
2. In the right pane, which now shows the current users of this database, right-click and select New Login. The New Login dialog will open.
3. Click the ellipses button (. . .) next to the Name box. This will open a window entitled SQL Server Login Properties - New Login. If it's not selected already, select your machine name from the List Names From drop-down list. In the Names list, scroll down the list of logins until you find the IUSR_MACHINENAME login (with the name of your computer in place of MACHINENAME). Select it and click Add.
4. Click OK, and the dialog will be filled out for you, showing the login details for IUSR. The name will be displayed as DOMAINNAME\IUSR_MACHINENAME (with the domain that your computer is a member of replacing DOMAINNAME, and the name of the machine replacing MACHINENAME). You don't want this login to be domain-specific, however, since you want people on the Internet to be able to use it, and they won't be logged in to your domain.
5. To change this, select the SQL Server Authentication radio button to make this login an SQL Server authenticated login.
6. Enter a strong password of your choosing—you'll need to remember this for use later on when you connect to the database from within Dreamweaver.
7. Select mysampledatabase in the Database drop-down list—this specifies the default database for the user you're creating.

Figure 4-8 shows the completed General tab of the dialog box.

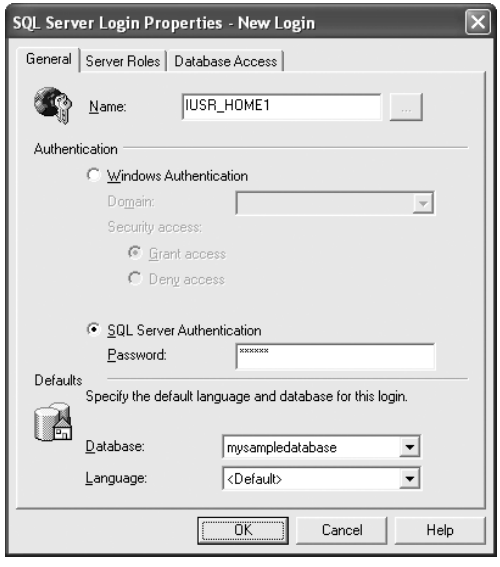


Figure 4-8. Adding a SQL Server login

8. Click the Database Access tab. Scroll down the list until you can see mysampledatabase. Tick the check box next to it, and your new login details will be added to that database (see Figure 4-9).

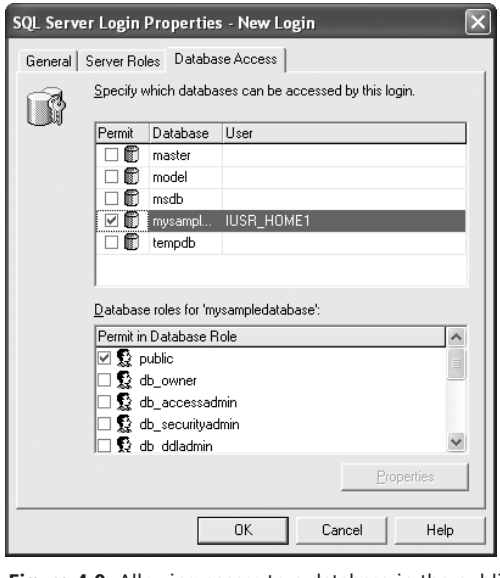


Figure 4-9. Allowing access to a database in the public user role

9. With that done, click OK. You'll then have to confirm the password for this new user in the Confirm Password dialog box (see Figure 4-10). Once that's confirmed, your new Internet user login will be created.

You can now set up a website that has the ability to connect to this database. You'll be doing exactly that later in this chapter, in the section entitled "Making The Connection."

Figure 4-10. Confirming the password to finish creating the new user

Database design

There's far more to good database design than just following a good naming convention. If you were to spend some time asking people their thoughts on the dos and don'ts, you would end up being able to fill an entire book. Lo and behold, someone beat you to it—perhaps one of the best books you can buy on the subject is *Database Design For Mere Mortals: A Hands-On Guide to Relational Database Design*, by Michael J. Hernandez (Addison-Wesley Professional, 1996). For a more specific title based around SQL Server, take a look at *Beginning SQL Server 2000 DBA: From Novice to Professional* (Apress, 2004).

I won't go into great depth here, but there are a couple of things you should consider when designing your database—if only to save you headaches later on!

Object naming conventions

SQL Server (and the SQL language itself) has a number of reserved keywords that must be avoided at all costs when creating names for your database objects and their properties. For a complete list, launch Books Online, go to the Index tab, enter keywords reserved for SQL Server, and click Display.

Using a good naming convention can help you to understand what's going on when you come to use your database later on, or when you return to a project you wrote months ago.

For example, let's say you have a few database tables, named as follows:

- `tblCustomers`
- `tblCustomer_Orders`
- `tblCustomer_Order_Details`
- `tblProducts`

It doesn't take a genius to figure out that `tblCustomer_Orders` would be a good place to look if you wanted to view some customer orders. Naming things logically makes it easy to quickly assess what an object contains and what it should be used for. This is especially useful when accessing these objects from external programs such as Dreamweaver.

In the preceding example, each table name is prefixed with `tbl`, which is short for "table." You do this so that you know it's a table and not a view, stored procedure, or any other database object. For views, I use a prefix of `view`, and for stored procedures, I use a prefix of `sp`.

Table column names are another area that can cause serious problems later on if you aren't careful. For example, say you have a table that contains customer orders and you want to include a column to store the date that an order was placed (i.e., the date it was added to the database). You shouldn't simply use `Date` as your column name, since `Date` is a reserved word in SQL Server. You can get around this by using `DateAdded` instead. This avoids any naming conflicts, and the column name still makes sense.

Avoid the use of spaces in object and property names. The examples shown here use the underscore character in place of spaces. You can use the capitalization method instead, for which the start of each new word is capitalized (instead of each word being separated with underscores). For example, using the capitalization method, the `tblCustomer_Orders` object would look like `tblCustomerOrders`.

Creating tables

A table structure defines the order and sequence in which data is stored and retrieved from your database. Each entry consists of a record, which you can think of as a horizontal row in your table, containing data in a series of columns, known as fields. Each field stores equivalent items of data for each record. It's common practice (and one that you'll adopt from here on) to make the first field a unique identifier for the record it belongs to—this unique identifier is known as a **primary key**. This field will be used later on when you begin to build relationships between the tables you've created.

The use of the primary key is summarized in the following table with two imaginary records.

Primary Key	Field 1	Field 2	Field 3	Field 4
1	Value for Field 1	Value for Field 2	Value for Field 3	Value for Field 4
2	Value for Field 1	Value for Field 2	Value for Field 3	Value for Field 4

Of course, you could store all the data for a given record in exactly the same form, and if you were dealing with a comma-delimited text file, that would be exactly what you would do. However, the values in each field of a record are almost always different in nature, depending on what each field is supposed to store, so the proper use of **data types** (such as `Integer` or `Date`) can greatly improve the efficiency of your database and help prevent inappropriate data being from entered in a given field. (You'll be looking at data types shortly.)

As an example, say you have an e-commerce site, and you're creating a table for it that stores all your customer details. In the table (let's call it `tblCustomers`), you'll have the following fields:

- A primary key, called `CustomerID`
- The name of the customer (be it an individual or business), in a field called `CustomerName`
- The customer's address, in a field called `PostalAddress`
- The customer's postal (zip) code, stored separately in a field called `Postcode`, since it can be used to quickly group customers by geographical region
- The customer's country of residence, `Country`—again, so that customers from a given country can be more easily grouped together

As you'll see, Access doesn't give you as much scope with regard to the data types as SQL Server does, but setting the correct data types for each of these fields and determining how much space each one can take up will help you maintain the integrity of the data, save on storage space, and even help to speed up data access.

Since data types are set at the table level, let's now take a look at the process of creating a table and the different data types that can be stored in the fields within it. You'll look at Access first, followed by SQL Server.

Creating tables in Access

Open the empty `mysampledatabase.mdb` database that you just created. Access 2003 had several security enhancements built in, so you'll be very likely to see the Security Warning (as shown in Figure 4-11) when you open your database—just click Open to continue.

Security Warning
Opening "C:\databases\mysampledatabase.mdb"
This file may not be safe if it contains code that was intended to harm your computer.
Do you want to open this file or cancel the operation?
Cancel
Open
More Info

Figure 4-11. Click Open to get past the security warning when opening a database in Access 2003.

With the database open, select Tables in the left column of the database window, click New, and select Design View from the pop-up dialog. (You can also double-click the Create Table in Design View icon in the Tables window).

You'll be presented with the table Design view, as shown in Figure 4-12.

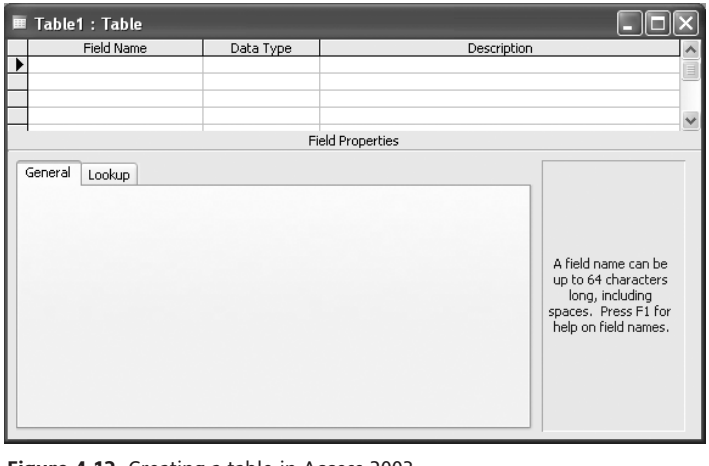


Figure 4-12. Creating a table in Access 2003

I've previously said that you can think of a record as rows of fields, running horizontally. You'll note, however, that you're stacking the field definitions vertically as you create them, which may seem a little counterintuitive at first. There are three values that you can assign to each field:

- Field Name: the name used by Access to identify a given field within a table.
- Data Type: a general class into which the data falls, (e.g., `Text`, `Number`, or `Currency`).
- Description: some text that you can use to remind yourself of what a given field is intended to hold. What you enter here has no wider use within the database itself, and is entirely optional.

Click inside the Data Type box for the topmost field and click the down arrow—you'll see a list of the general data types that Access recognizes. If you select a general data type, such as `Number`, for example, and then move down to the Field Properties dialog and select Format (for most of the general types) or Field Size (in the case of `Number` data types), you'll see another down arrow appear. Click this and you'll see all of the subtypes you can assign to your data. For instance, the subtypes of the `Number` data type are as follows:

- `Byte`
- `Integer`
- `Long Integer`
- `Single`
- `Double`
- `Replication ID`
- `Decimal`

If you've done any programming before, you may well be familiar with the concept of using different data types to optimize how your data is stored and handled within your code. If not, don't worry, you can play around with the different data types and field lengths at your leisure—most of them are fairly easy to grasp—and you won't need a thorough grounding in this for the simple table you're about to create.

Let's now take a look at creating the previously mentioned `tblCustomers` table in Access.

To create the primary key, type CustomerID in the first Field Name box in Design view. Give it a data type of `Autonumber`. Access will now ensure that the value created here for each record will be unique. The field size will be set as Long Integer by default, so leave it like that. Now click the gray square to the left of the Field Name column to highlight the entire row. Right-click, and you'll see that the first entry in the pop-up menu that appears is Primary Key. Select this, and a small key icon will appear in the gray square to the left of the field name. This signifies that Access will now treat this field as the primary key for this table. To maintain integrity, Access allows you to assign this property to only one field in a table at any one time, although you can select multiple rows to assign the primary key to a combination of columns—but you don't need to do this here.

Move down to the next field and name it CustomerName. Give it a data type of `Text`. You may wish to increase the Field Size (the number of individual characters) to 100 or so, from the default value of 50. In order to avoid wasting space, always aim to make a field no bigger than the maximum amount of data you expect it to have to hold. However, you should bear in mind that if you try to insert more data than a column is designed to hold, you'll get an error and the insert will fail.

Name the next field PostalAddress, and give it a Field Type of Text and a size of 250. Be aware that text fields can hold a maximum of 255 characters anyway, so if you feel this field should be able to hold more characters than this, you'll need to use a Memo data type. For this example, however, use a Text type to keep things simple.

Name the next field Postcode. Give it a Field Type of Text. A Field Size of 10 should be sufficient.

Name the next field Country, and give it a Field Type of Text and a Field Size of 50.

You save your newly created table by clicking the Save icon, or by selecting the File option from the menu, choosing Save As, and entering tblCustomers as your new table's name. You can now view your table in Datasheet view by selecting View ➤ Datasheet View. If you had simply tried viewing your table in Datasheet view straightaway, Access would have prompted you to save you table first. Either way, you've now created your new `tblCustomers` table.

Creating tables in SQL Server

In the left pane under Databases, select the empty database called mysampledatabase that you created earlier. In the right-hand pane, right-click the Tables icon and select New Table. The New Table dialog box will open.

When you enter the field name in the Field Name column, and then move into the Data Type column and select the down arrow that will appear, you'll then see a list of all the data types that SQL Server supports. Unlike Access, SQL Server's data types don't break down into more specific subtypes; also, SQL Server has a great many more actual types than Access does. This reflects the fact that SQL Server is optimized for fast data retrieval and efficiency of storage.

If you refer back to your example table, `tblCustomers`, you'll recall that it has a primary key field named `CustomerID`. Figure 4-13 shows how your New Table dialog box looks just after you've created this field.

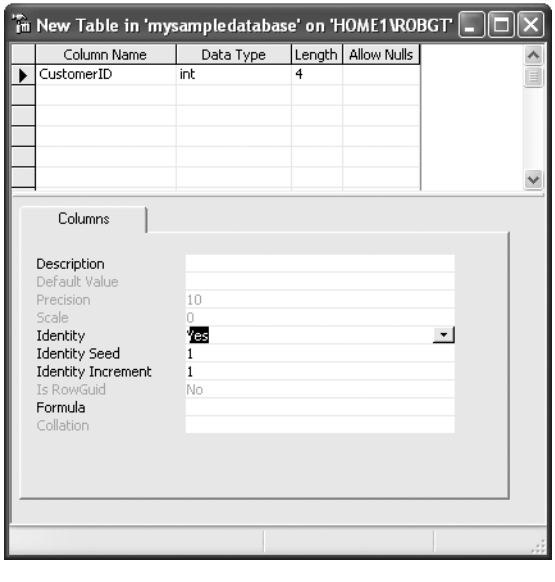


Figure 4-13. The New Table dialog box, in which the CustomerID column is chosen as an Identity column

As shown in Figure 4-13, give this field a name of CustomerID and a data type of int. Then set the Identity box in the Columns tab to Yes. Identity is the name for SQL Server's built-in primary key data type. This defaults to an identity seed (the number from which the identity field will begin counting when the first record is created) and identity increment (the rate at which the value is increased with each new record) of 1 in each case. This means that the first record will be given a primary key of 1, and each successive record will have a primary key one greater. This is fine for your purposes, but SQL Server does provide the ability to customize many elements of how your data should be stored.

The int data type occupies 4 bytes by default. You'll find that for this reason you can't change the value in the Length column. If this were a text field, such as a char (character) type, you could set the number characters it occupies and, thus, what kind of load on the total storage space the field would exert for each record. As you saw when carrying out this process in Access, the decision about how big a field should be must be taken with care. An int data type can store up to around 2 billion values, which may seem like overkill (even if this were Amazon.com that you were designing for)—but remember that any value stored here can never be used again in the same table, and customers will come and go. In the lifetime of your database, the use of an int data type for a field of this kind is appropriate—especially since the next smallest size available is tinyint, which can store only up to 256 different values.

Now you can create the other fields: CustomerName, PostalAddress, Postcode, and Country. For normal fields like this, you can permit the value to be a null, which is literally an unknown value. It's not an empty string—that is, a character string with no content (""), which contains nothing but is still something—it's just an empty something. The use of a null explicitly stores the fact that no value exists. You can also specify a default value, which is a value that will always be entered in a field, should no other value be provided in its place.

Create CustomerName as a char data type of length 100. You may think this is too much, but it depends on whether you would expect to be receiving commercial customers or not. A char data type is of fixed length, which is to say that its field will always hold 100 characters, even if its customer name were simply John Smith—the rest of the field would be filled up with spaces. This may sound very inefficient, and you could have made this field a varchar data type instead, meaning that each field for a given record would occupy only as many characters as are needed to store its value. However, the use of fixed-length fields allows SQL Server to search for and retrieve values much faster, since it doesn't have to check for the end of each field all the time. Make sure you uncheck the Allow Nulls column, and don't bother entering any default values, since neither of these options is appropriate for this field.

Now create a field named PostalAddress with a data type of char, and give it a length of 250 characters. Make both Postcode and Country fields of type char, with lengths of 10 and 50 characters, respectively. If this were a real example, you might want to assign Country with a default value—if you expected the majority of your customers to come from your home country, for instance.

Your finished New Table dialog should look like Figure 4-14.

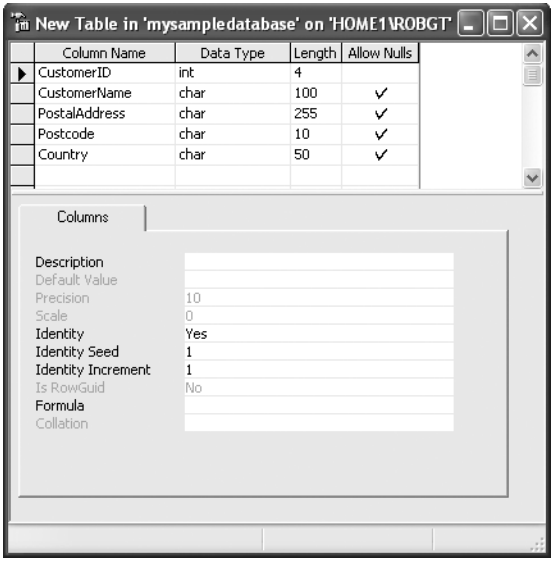


Figure 4-14. A completed SQL Server table design

Save this table by clicking the Save icon in the top-left of the main window; name it `tblCustomers`. Close the dialog and click the Tables icon in the right-hand pane. Your new `tblCustomers` table will be listed in the resulting display.

Relational databases and referential integrity

Relational databases allow you to store lots of information in a structured and organized way, removing the need to store multiple copies of the same data. The use of primary and foreign keys is crucial to how you achieve this.

For example, in your e-commerce site, suppose a customer places an order for five items. You would want to store the customer's delivery details, an overview of their order, and the details of their order (the individual items they bought).

You could store all this information in one table and have the customer's name and address entered with each record of the order, but that's a lot of unnecessary repetition. Fortunately, thanks to relational databases, you can split the information over several tables.

- The `tblCustomers` table you've just created would have just one record per customer and would store the contact and delivery details.
- The `tblCustomer_Orders` table would have one record per order and might detail the order total, the date and time the order was placed, and whether the order has been fulfilled or not.
- The `tblCustomer_Order_Details` table would hold a record of each individual item ordered (the details of the order).

In this case, you would set up two relationships to show how the entries relate to each other: one would be between tblCustomers and tblCustomer_Orders, and the other would be between tblCustomer_Orders and tblCustomer_Order_Details. You do this using the value of CustomerID, the primary key from the tblCustomers table. The tblCustomer_Orders and tblCustomer_Order_Details tables will each contain a field that stores the CustomerID values of the customers who placed these orders—the same CustomerID values you stored in tblCustomers. This is known as a **foreign key**. A foreign key is simply a primary key from one table recorded as a field within another table.

Like a leash connects a dog to its master, the foreign key in the tblCustomer_Orders and tblCustomer_Order_Details tables ties the order records to the customers to whom they belong—from this, you can retrieve the details of each customer, along with their orders.

So, in tblCustomer_Orders, you would have a column called CustomerID that holds the foreign key value of CustomerID for the relevant record in tblCustomers. You would also have a column called OrderID in the tblCustomer_Order_Details table that holds the foreign key value of OrderID for the relevant record in tblCustomerOrders.

> *These foreign key columns aren't* Identity *or* Autonumber *columns (which contain unique values only) because you might have to store several records with the same* CustomerID *values in the* tblCustomer_Orders *table—or, even more likely for single orders containing multiple products, several records with the same* OrderID *values in the* tblCustomer_Order_Details *table.*

So let's recap. Using the example of a customer buying five items from your store, there would be one record created in tblCustomers, one record created in tblCustomer_Orders, and five records created in tblCustomer_Order_Details. The one tblCustomer_Orders record would contain a primary key of its own, called OrderID, and a foreign key from the tblCustomers table—CustomerID. The five tblCustomer_Order_Details records would each contain their own primary keys called OrderDetailsID, for example, and also a single OrderID foreign key that links them to the order that they were part of.

Let's see how this is done. You might want to create some dummy tables called tblCustomer_Orders and tblCustomer_Order_Details, so that you can try this next example out for yourself. You can give each one dummy fields to reflect how you think they would be structured, if you like. However, all you really need is a primary key for both OrderID and OrderDetailsID, a field called CustomerID in the tblCustomer_Orders table, and a field called OrderID in the tblCustomer_Order_Details table. These latter two fields will be the foreign key in each case.

Creating a relationship in Access

To create a relationship in Access, click the relationships button on the toolbar (see Figure 4-15).

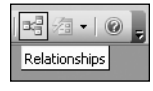


Figure 4-15.
The Relationships toolbar button in Access 2003

You'll now be looking at a large, blank gray canvas onto which you need to add the tables you want to relate to each other.

Let's follow on from the examples shown before and relate the `tblCustomers` table to a `tblCustomer_Orders` table, which would be the first of two relationships you would set up if you were to follow the examples through to their conclusion.

Right-click and select Show Table from the context menu. From the dialog, select `tblCustomers` and click Add. You should also add `tblCustomer_Orders` and `tblCustomer_Order_Details`.

In the list of columns in the `tblCustomers` object, left-click and drag from the `CustomerID` field to the `CustomerID` field in the `tblCustomer_Orders` object, and then release the mouse button. As you start to drag between the objects, the cursor changes to a small horizontal bar. When you've finished dragging and have dropped (released the click you started on `tblCustomers`), the Edit Relationships dialog box opens, which allows you to specify the details of the relationship (see Figure 4-16).

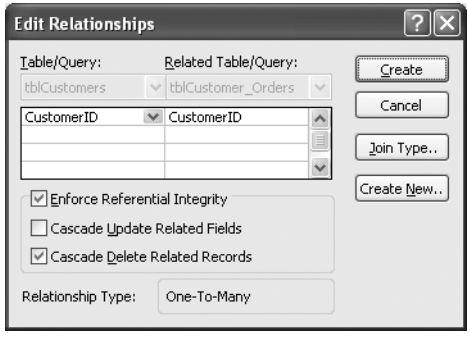


Figure 4-16. The Edit Relationships dialog box

You want to set up a **one-to-many** relationship between `tblCustomers` and `tblCustomer_Orders`. The columns that have the related data in them will both be called `CustomerID` in this case. They don't have to be named the same; they just have to contain the same data and be stored using the same data type. But you should use the same name for simplicity's sake—simple is good!

As shown previously, you also want to specify Cascade Delete Related Records in the referential integrity area so that if a record is deleted from my `tblCustomers` table, all the related records from my `tblCustomer_Orders` table will be deleted as well.

Click Create to create the relationship, and a black line will be drawn between the two tables to signify the relationship. Notice that there's a 1 above the end of the line that ends at `tblCustomers`, and an infinity symbol (which looks like a sideways 8) above the other end. These visually signify the one-to-many relationship you've created between these two tables.

To complete the three-table relationship, perform the same process between the `tblCustomer_Orders` and `tblCustomer_Order_Details` tables, using the `OrderID` field in both tables.

The completed relationship view will look like Figure 4-17.

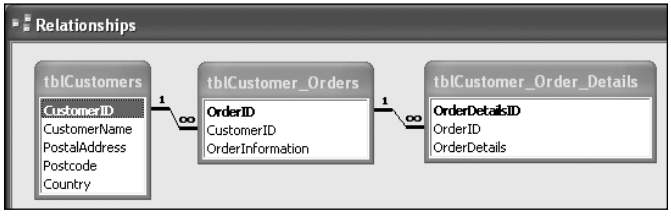


Figure 4-17. The Relationships window showing three related tables

Creating a relationship in SQL Server

As ever, with SQL Server, there's more than one way to achieve your objective. To design your relationships in a non-visual way, you can, from the table Design view, click Manage Relationships on the toolbar and use the resulting dialog to create them. You can alternatively go though the visual process of creating a Database Diagram, as is about to be discussed.

In the left pane of Enterprise Manager, expand the `mysampledatabase` database, and then click the `Diagrams` object.

Right-click in the right pane and select New Database Diagram from the context menu to launch the Create Database Diagram Wizard. Click Next.

Now you can select the tables to add to your diagram by double-clicking them, or by clicking on them and clicking the Add button, located between the left and right lists. If you check the Add related tables automatically option, it will automatically add any tables to the diagram that are already related to the table you wish to add. You can also specify how many levels of related tables it should add automatically. The default is 1. You want to add `tblCustomers`, `tblCustomer_Orders`, and `tblCustomer_Order_Details` to your diagram, so select them and click Add to put them into the list on the right (as shown in Figure 4-18).

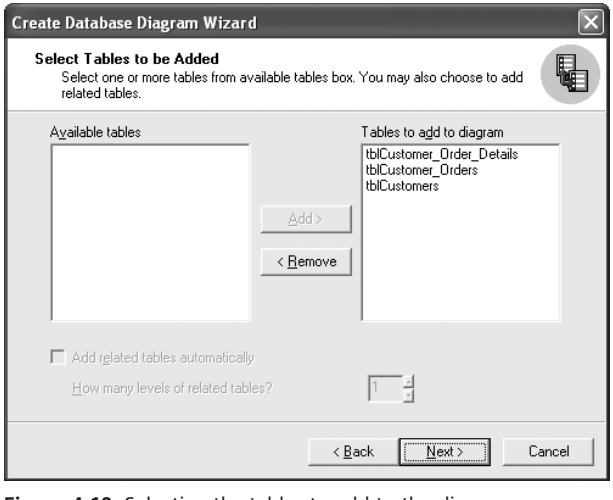


Figure 4-18. Selecting the tables to add to the diagram

Click Next to check the details, and click Finish if everything looks OK. (If you make a mistake, click Back to go back and correct it.) SQL Server will now generate a basic diagram for you.

This now works in pretty much the same way as Access. Once again, from the list of fields in the `tblCustomers` object, left-click and drag from the `CustomerID` field to the `CustomerID` field in the `tblCustomer_Orders` object, and release. A dotted line is drawn between the objects you're connecting as you do so. When you drop the drag, the Edit Relationships dialog opens up for you to specify the details of this relationship (see Figure 4-19).

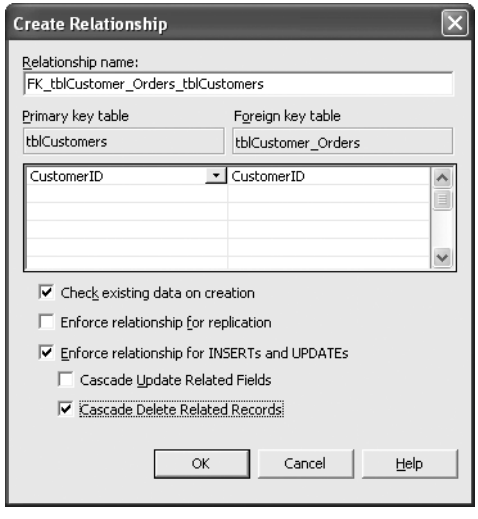


Figure 4-19. The Create Relationship dialog box

One of the very useful options here is the first check box, Check existing data on creation. This allows you to ensure that any data already in these tables conforms to this relationship. If you know it doesn't, make sure this check box is left unchecked. When creating relationships on empty tables, it doesn't matter if you leave it checked or not, of course.

If you plan to use this database in a replication environment (i.e., a collection of databases that synchronize their content with each other at predefined intervals), you can specify that this relationship should be enforced by checking the second check box, Enforce relationship for replication. You aren't going to be getting involved with replication in this book, so you can leave that one unchecked.

Finally, you have the Cascade Update Related Fields and Cascade Delete Related Records options. You can deselect the update option since it won't affect this relationship in this instance. You can't change the `CustomerID` value for a record in your `tblCustomers` table because it's the `Identity` value, which, for all intents and purposes, is a read-only value.

Click OK to create the relationship.

The diagram now shows a line with a gold key at one end and an infinity symbol at the other to visually signify the one-to-many relationship you've created between these two tables.

To complete this three-tier relationship, you can create the second relationship between `tblCustomer_Orders` and `tblCustomer_Order_Details`, linking the `OrderID` field in each table. With that done, the diagram should look like Figure 4-20.

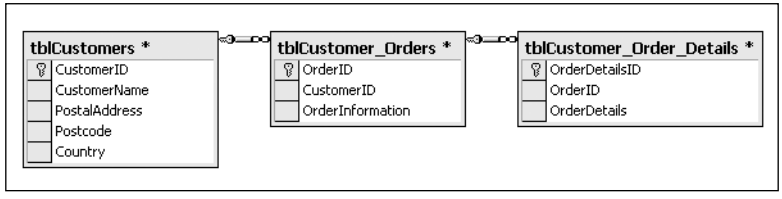


Figure 4-20. The three-table relationship completed

You then need to save the diagram. You can use the default name of `Diagram 1` if you want to, although a descriptive name may help you in the future if you create many diagrams in your database. Try something like `Customer Orders`. (Spaces in diagram names are fine.)

When you save your diagram, some behind-the-scenes modifications need to be applied to the tables involved, to ensure that the relationships you just specified will be adhered to. Click Yes in the dialog that appears to allow these changes to be saved.

SQL Server views and Access queries

SQL Server **views** and Access **queries** are the ideal way to bring together the data you need to retrieve from the tables in your database.

Using views or queries in your database allows you to store some of the SQL work within your database rather than having to pass an entire statement to the database from your web page. This means that to change the data displayed on a page, you might only need to modify the code in your database, rather than editing the SQL in your web page.

The need to build very complex SQL queries can rear its head quite quickly in large projects, which is why I almost always use views or queries to gather the data I want, and then use a simple select statement to grab all of those records in my web page. I may even pass filtering parameters into the view to narrow down my recordset even further. The beauty of doing it this way is that the complex SQL statement stays in the database, and the runtime performance of the queries can be increased so that your web pages will load faster.

Say you want to view a record of sales by seeing who has bought products from your online store, along with how much they spent in total. You don't need to know what they bought or where they live at this point, you only want to see the crucial data—their name and their total purchase cost.

To generate the recordset for a web page to show you this data in Dreamweaver, you could open the Recordset builder, switch to Advanced View, and build up the following statement:

```
SELECT dbo.tblCustomer_Orders.OrderID, dbo.tblCustomer_Orders_FullName,
       SUM(dbo.tblCustomer_Order_Details.TotalCost)
AS TotalPurchaseCost
FROM dbo.tblCustomer_Orders INNER JOIN dbo.tblCustomer_Order_Details
ON dbo.tblCustomer_Orders.OrderID =
  dbo.tblCustomer_Order_Details.OrderID

GROUP BY dbo.tblCustomer_Orsders.OrderID,
   ➥dbo.tblCustomer_Orders_FullName
```

The Recordset builder is great in that it can get you a long way toward creating this SQL statement. It won't be able to do it all, however, so you would have to add some of this code by hand. As I've said before, it's often easier to get visual tools to do the job for you. You'll be looking at how to use the **Simple Recordset builder** and the **Advanced Recordset builder** later in this chapter.

A simpler way to do this would be to create a view called `viewPurchase_Totals` in SQL Server (or a query in Access) to house the preceding SQL code, and then to use the Dreamweaver MX Recordset builder to build a recordset with the following code:

```
SELECT * FROM view_Purchase_Totals
```

I know which I'd prefer!

Views and queries become increasingly useful when you need to gather data from more and more tables. Using a view or query to select records from a single table may offer only slight benefits, and only if the table is large, if it has many columns, and if you're selecting most of them. If it's a small table, then the benefits are often much smaller.

Fundamental SQL

Later in this chapter, you'll see just how easy Dreamweaver MX makes building SQL statements to create recordsets for your web pages. To give you a full understanding of this creation process and the final SQL code that it produces, a quick primer in some fundamental SQL is in order.

All the SQL statements you'll see here can be used in Dreamweaver's Advanced Recordset builder. Some of them can be created in the Simple Recordset builder, too—the difference being that when you need to specify more than one criterion in your statement (a `WHERE` statement that needs to meet two or more criteria), you'll need to use the Advanced Recordset builder.

SQL Server uses a strict naming convention for its objects. If you have programming experience of a language that uses dot notation, then you'll recognize this and understand it easily. If not, it really is quite logical.

To reference an object in SQL Server, you use the following standard:

```
databaseowner.objectname
```

In the current example, this code in action would look like the following:

```
dbo.tblCustomer_Orders
```

> *Access doesn't need this extra owner information, nor would it understand it if you used it. The reason is that Access doesn't use the ownership metaphor or permissions principles that SQL Server does. It's intended as a single-user database manager. SQL Server ensures that only the people with the relevant permissions can perform actions on the database objects; in Access, it's a free-for-all!*

The following examples assume that the owner of the database objects in SQL Server is dbo. Also, you'll be addressing the same imaginary table object, tblLogins, in each case. The SQL code for each example is shown for both Access and SQL Server.

Selecting all records from a table

This is probably the simplest SQL statement there is.

Access:

```
SELECT * FROM tblLogins
```

SQL Server:

```
SELECT * FROM dbo.tblLogins
```

This statement says "Select all the records contained in tblLogins." Here, * means all columns. This is not to be confused with the mathematical *, which looks the same but performs a very different role (i.e., multiplication).

Selecting all records that meet one criterion

This is almost as easy as the first example, but there are a couple of crucial rules to bear in mind. You're going to use a similar statement to the one preceding, but you only want to return those records that contain a specific value in the specified column. You use the keyword WHERE to specify this.

Access:

```
SELECT * FROM tblLogins WHERE Username = 'value'
```

SQL Server:

```
SELECT * FROM dbo.tblLogins WHERE Username = 'value'
```

Strikingly similar, aren't they?

This statement says "Select all the records contained in tblLogins where the value contained in the Username column matches the value specified (in place of value)."

What you may need to watch out for in SQL statements is the data type of the column that stores the value you need to match. In the previous example, the column Username has a text-based data type, which means that the value you're going to compare with the value in that column needs to be enclosed in single quotes. If it had been a numeric column, single quotes do not need to be used.

To illustrate this point, the following SQL code shows you the correct way to use a numeric value in the value area of this statement.

Access:

```
SELECT * FROM tblLogins WHERE LoginID = value
```

SQL Server:

```
SELECT * FROM dbo.tblLogins WHERE LoginID = value
```

This statement says "Select all the records contained in tblLogins where the value contained in the LoginID column matches the value specified (in place of value)."

The LoginID in your database is a numeric value; therefore, the comparison value needs to be numeric too. By removing the single quotes from around the value, the SQL Statement becomes valid. If you left the single quotes around the value and tried to execute the statement, you would get a data type mismatch error because the SQL code sees that you're trying to use a non-numeric value in a comparison statement with the value contained in a numeric column.

Bearing these facts in mind, let's quickly step through a couple of increasingly complex, yet still very straightforward, examples.

Selecting all records that meet several criteria (using AND)

The AND keyword allows you to create SQL statements that specify that more than one criteria must be met in order to return any results.

Access:

```
SELECT * FROM tblLogins WHERE Username = 'value'
  AND Password = 'anothervalue'
```

SQL Server:

```
SELECT * FROM dbo.tblLogins WHERE Username = 'value'
  AND Password = 'anothervalue'
```

This statement says "Select all the records contained in tblLogins where the value contained in the Username column matches the first value specified (in place of value), *and* the value contained in the Password column matches the second value specified (in place of anothervalue)."

Selecting records that meet one or more of several criteria (using OR)

The OR keyword allows you to create SQL statements that specify that one or more criteria of *either* of the specified criteria must be met in order to return any results.

Access:

```
SELECT * FROM tblLogins WHERE Username = 'value'
  OR Password = 'anothervalue'
```

SQL Server:

```
SELECT * FROM dbo.tblLogins WHERE Username = 'value'
  OR Password = 'anothervalue'
```

This statement says "Select all columns for all the records contained in `tblLogins` where the value contained in the `Username` column matches the value specified (in place of `value`), *or* the value contained in the `Password` column matches the value specified (in place of `anothervalue`)."

If you were to build a slightly more complex SQL statement that utilizes the `OR` keyword *and* the `AND` keyword, you would be well advised to use parentheses to encapsulate the `OR` criteria. If you don't do this, you may get erroneous results. The following Access-based examples will explain this. To use this example in SQL Server, just add `dbo.` in front of the table name.

```
SELECT * FROM tblLogins
WHERE (Username = 'value' OR Password = 'anothervalue')
  AND AccessLevel = 'yetanothervalue'
```

This would correctly return all columns for all records from the `tblLogins` table that meet the criteria of having either a matching `Username` value *or* a matching `Password` value, *and* having a matching `AccessLevel` value. Compare it with the following statement, which could return incorrect results:

```
SELECT * FROM tblLogins
WHERE Username = 'value'
  OR Password = 'anothervalue' AND AccessLevel = 'yetanothervalue'
```

Here, the SQL might be saying what the first example said, or it might be interpreted as saying "Return all records from the `tblLogins` table that meet the criteria of having a matching `Username` value, *and* also having a matching `AccessLevel` value *or* a matching `Password` value." It's ambiguous. If you say the whole statement out loud with a big emphasis on the `OR` keyword, you'll understand the point I'm making here.

Basically, the more complex your SQL statements become, the more careful you need to be in creating them.

Useful SQL keywords

There are far too many SQL keywords for me to cover in this section of the book. However, a quick glance at some of the more commonly used ones might help you to understand how to achieve the results you're after. To see all the available SQL keywords, be sure to check out Books Online.

COUNT

If you want to count how many login records you've stored in your tblLogins table, for example, you could use the COUNT keyword in the following way.

Access:

```
SELECT COUNT(LoginIncluded) AS TotalLogins FROM tblLogins
```

SQL Server:

```
SELECT COUNT(LoginIncluded) AS TotalLogins FROM dbo.tblLogins
```

This actually illustrates two very useful SQL keywords in one: COUNT and AS.

The COUNT(LoginIncluded) section will count how many rows there are in the tblLogins table, while the AS TotalLogins section will return the COUNT value to an alias column called TotalLogins. TotalLogins doesn't exist as a column in your database, but you're declaring it as an alias by using the AS keyword.

The preceding SQL statement might be more useful if you used a selection criteria to count how many rows there are in your tblLogins table that are included—this would be signified by a True value in the LoginIncluded column, as follows:

Access:

```
SELECT COUNT(LoginIncluded) AS TotalLogins
  FROM tblLogins WHERE LoginIncluded = -1
```

SQL Server:

```
SELECT COUNT(LoginIncluded) AS TotalLogins
  FROM dbo.tblLogins WHERE LoginIncluded = 1
```

SUM

If you want to add up the values of several rows (for example, to find the total value of a customer's orders) to create a grand total, you could use the SUM keyword in your SQL statement, as in the following example.

Access:

```
SELECT SUM(TotalCost) AS GrandTotal
  FROM tblCustomer_Order_Details WHERE OrderID = 2
```

SQL Server:

```
SELECT SUM(TotalCost) AS GrandTotal
  FROM dbo.tblCustomer_Order_Details WHERE OrderID = 2
```

The SUM(TotalCost) section will add together all the rows in the tblLogins table that match the WHERE criteria and return a single row contained in a column called GrandTotal, using the alias keyword AS. In this example, the rows that match must have an OrderID of 2.

TOP

The TOP keyword is very useful if you want to specify that only a certain number of rows should be returned to your recordset. Obtaining exactly the amount of data you require is good for performance reasons.

For example, if you have a table that contains hundreds of thousands of rows of DVD and CD products, and you know that over 10,000 of those rows will match your specific WHERE criteria, but you only want to view a sample of these records, then the TOP keyword can arrange that for you.

All you need to decide is how many records you want to retrieve. Let's say that you want 100. Following is the code that you would use.

Access:

```
SELECT TOP 100 * FROM tblProducts WHERE ProductCategory = 'DVD'
```

SQL Server:

```
SELECT TOP 100 * FROM dbo.tblProducts WHERE ProductCategory = 'DVD'
```

The process this statement goes through is fairly straightforward. It filters the tblProducts table with the criteria you specified (ProductCategory must equal DVD); then, from that result set, it selects the first (TOP) 100 records.

Notice the * immediately after the 100 figure in the statement. This signifies which columns of the table you want returned. You could have specified one or more column names here to narrow down even further the data that's returned—but in this case, all of the columns were desired, so * was used.

This is all well and good, but what if you don't want the top 100 records? Say you wanted to view the last 100 DVDs added into your product database between two dates. To do that, you could either specify the dates in your SQL statement using the BETWEEN keyword, or you could use the ORDER BY keyword. You'll look at both of these in turn.

BETWEEN

The BETWEEN statement has a lot of uses, and just a couple of simple stipulations on how it's used. Let's look at an example of BETWEEN in use, and then break it down to see what's going on.

Access:

```
SELECT TOP 100 * FROM tblProducts
WHERE ProductCategory = 'DVD'
  AND DateAdded BETWEEN #01 August 2005# AND #31 August 2005#
```

SQL Server:

```
SELECT TOP 100 * FROM dbo.tblProducts
WHERE ProductCategory = 'DVD'
  AND DateAdded BETWEEN '01 August 2005' AND '31 August 2005'
```

In the preceding code, you want to select all columns of data for the TOP 100 records that are returned that match the criteria specified—in this case, the ProductCategory is DVD, and the DateAdded column must contain a date that's *between* the dates of the 1st and the 31st of August 2005, inclusive.

There are a couple of things that you need to keep an eye out for when using BETWEEN:

- It must specify inclusive values.
- It must be compared to the same data type.

In the preceding example, DateAdded has a Date data type, so the first and second values must be Date values also.

For a slightly more in-depth look at dates and how they're used in SQL, see the "Going on a DATE" section later in this chapter.

ORDER BY

ORDER BY is a powerful keyword that you can use to order the way your records are returned to you. For example, you might want them to be in the order they were entered into the database, using the primary key value, which is usually a sequential number for each row. You might also want to specify multiple columns by which to order your data. The way in which data is returned to you is dependent on the order in which you specify the columns in the ORDER BY statement.

A couple of examples will help to illustrate this powerful keyword.

Access:

```
SELECT TOP 100 * FROM tblProducts
WHERE ProductCategory = 'DVD'
  AND DateAdded BETWEEN #01 August 2005# AND #31 August 2005#
ORDER BY ProductCategory, ProductPrice
```

SQL Server:

```
SELECT TOP 100 * FROM dbo.tblProducts
WHERE ProductCategory = 'DVD'
  AND DateAdded BETWEEN '01 August 2005' AND '31 August 2005'
ORDER BY ProductCategory, ProductPrice
```

Using the same example as the BETWEEN example, the ORDER BY keyword has been added so that all the results that meet the criteria specified will be ordered by their ProductCategory; and within that, by their ProductPrice.

This means that the DVDs that match the SQL statements entire criteria will be returned, ordered such that the cheapest DVD from the first category will be listed first, followed by increasingly expensive DVDs from the first category, followed by the cheapest DVD from the second category, followed by increasingly expensive DVDs from the second category, and so on—until all the matching records have been sorted into this order. Then the TOP 100 DVDs *from that set* will be retrieved.

You can also specify the direction in which a sort order is used, either from the smallest value to the largest value, or from the largest to the smallest. This is achieved by using the ASC or DESC keywords (short for ascending and descending).

The default sort order is ascending; you can use it in your SQL statements, but it's not necessary unless you're specifying more than one sort order.

To sort the previous example so that the most expensive DVDs are listed first and the cheapest are listed last within each category, but the categories remain in ascending order, you can add the ASC and DESC keywords to the statement, as in the following example.

Access:

```
SELECT TOP 100 * FROM tblProducts
WHERE ProductCategory = 'DVD'
  AND DateAdded BETWEEN #01 August 2005# AND #31 August 2005#
ORDER BY ProductCategory ASC, ProductPrice DESC
```

SQL Server:

```
SELECT TOP 100 * FROM dbo.tblProducts
WHERE ProductCategory = 'DVD'
  AND DateAdded BETWEEN '01 August 2005' AND '31 August 2005'
ORDER BY ProductCategory ASC, ProductPrice DESC
```

IN

The IN keyword is useful for those times when you might have several criteria that you want to use as a filter on your data.

You could write a long WHERE clause that uses many OR statements, or you could write a much shorter WHERE clause that uses the IN statement. The following examples illustrate this point.

First, have a look at using the OR keyword.

Access:

```
SELECT ProductName, ProductCategory,
  ProductDescription, ProductPrice
FROM tblProducts
WHERE ProductCategory = 'Cat 1' OR ProductCategory = 'Cat 2'
  OR ProductCategory = 'Cat 3' OR ProductCategory = 'Cat 4'
  OR ProductCategory = 'Cat 5'
```

SQL Server:

```
SELECT ProductName, ProductCategory,
  ProductDescription, ProductPrice
FROM dbo.tblProducts
WHERE ProductCategory = 'Cat 1' OR ProductCategory = 'Cat 2'
  OR ProductCategory = 'Cat 3' OR ProductCategory = 'Cat 4'
  OR ProductCategory = 'Cat 5'
```

Now, look at how this could be written using the IN keyword. You specify the values to compare against the column within parenthesis.

Access:

```
SELECT ProductName, ProductCategory,
  ProductDescription, ProductPrice
FROM tblProducts
WHERE ProductCategory IN ('Cat 1','Cat 2','Cat 3','Cat 4','Cat 5')
```

SQL Server:

```
SELECT ProductName, ProductCategory,
  ProductDescription, ProductPrice
FROM dbo.tblProducts
WHERE ProductCategory IN ('Cat 1','Cat 2','Cat 3','Cat 4','Cat 5')
```

If the data type that you're comparing a value against is numeric, it doesn't require single quotes around it. The preceding example uses them because the ProductCategory column is a text-based column.

GROUP BY

The GROUP BY keyword groups identical values in a column in your result set. In this regard, it's quite similar to the ORDER BY keyword. It differs, however, in that it doesn't allow certain data types to be specified in the overall SQL statement. For example, a Memo field in Access, or its equivalent Text field in SQL Server, can't be used in a GROUP BY statement (unless they're being used as an expression of criteria to be met—for example, if you simply want all the records with a description column that's not empty).

When you use GROUP BY, data will be sorted first in the order of the columns specified in the GROUP BY statement, then in an ascending order, row by row, unless you also specify an ORDER BY statement as well.

For example, in the tblProducts table, you might have many products that share the same category, and you might want to return a result set that groups together the results by category and then by price, as in the previous ORDER BY example.

A valid GROUP BY statement might be created as follows.

Access:

```
SELECT TOP 100 ProductName, ProductCategory, ProductPrice
FROM tblProducts GROUP BY ProductName, ProductCategory, ProductPrice
```

SQL Server:

```
SELECT TOP 100 ProductName, ProductCategory, ProductPrice
FROM dbo.tblProducts
GROUP BY ProductName, ProductCategory, ProductPrice
```

You can see that the columns that you want to be returned by this statement have to be specified. You can't use the * wildcard to return all columns because you can't include all the columns of this table in a `GROUP BY` clause.

The following example shows the use of the `GROUP BY` statement in conjunction with the use of a column that contains data that can't be returned to the result set, but that can be used to filter that result set.

Access:

```
SELECT TOP 100 ProductName, ProductCategory, ProductPrice
FROM tblProducts
WHERE (ProductDescription IS NOT NULL)
GROUP BY ProductName, ProductCategory, ProductPrice
```

SQL Server:

```
SELECT TOP 100 ProductName, ProductCategory, ProductPrice
FROM dbo.tblProducts
WHERE (ProductDescription IS NOT NULL)
GROUP BY ProductName, ProductCategory, ProductPrice
```

You still select the same columns as before, but the results will be filtered to show only records that have a value that's *not null* in the `ProductDescription` column.

The use of the `NOT` keyword comes into play here. It's used to reverse the logic of the statement it's contained within. In this case, you reversed the `IS NULL` statement by specifying `IS NOT NULL`, so that all records that have a `ProductDescription` will be returned.

DISTINCT

`DISTINCT` is used to prevent duplicate rows from being returned to the result set.

If you absolutely must have a result set that contains at least one unique value on every row, then `DISTINCT` can be the answer. However, it isn't always necessary to use `DISTINCT` because, generally, you'll return the primary key value of each record in your query. If the need arises for a result set that doesn't include the primary key value, then `DISTINCT` will ensure that no duplicate records are returned.

The following is an example of using the `DISTINCT` keyword.

Access:

```
SELECT DISTINCT ProductCategory FROM tblProducts
```

SQL Server:

```
SELECT DISTINCT ProductCategory FROM dbo.tblProducts
```

This will select all the individual categories from the `ProductCategory` column in the `tblProducts` table. You may have thousands of rows of products in that table; each of them stored in, let's say, 20 categories. This example will return those 20 distinct rows to your result set.

Going on a DATE

Dates are notoriously awkward when it comes to databases. If you live anywhere outside of the countries that follow the US date format, you can find yourself in all sorts of trouble when storing dates in your database.

In the examples shown for the keyword `BETWEEN` earlier in this chapter, the date was specified in a manner that leaves no ambiguity over how it should be read. However, depending on where you are in the world, 01/08/2005 could mean the first of August or the eighth of January.

Access and SQL Server always try to store the date in the US format: month/day/year. If they come across a date that can't be stored this way, such as 17/08/2005, then it's converted into the month/day/year format, while keeping the date accurate. To save yourself a lot of trouble, just use unambiguous dates like 17 August 2005.

When using dates in your SQL statements (such as when you perform a search for matching items that have a date that falls between two dates that you specify), you'll need to remember the following strict rules about how each database application needs dates presented to it:

- Access needs the date value to be wrapped in hash (or pound) symbols; for example, `#01 August 2005#`.
- SQL Server needs the date value to be wrapped in single quotes; for example, `'01 August 2005'`.

Making the connection

To be able to use data on your web pages, you need to tell your web application where to look for it.

There are two ways of doing this: you can either tell Dreamweaver that a system DSN (data source name) has been set up on your computer and you wish to use that, or you can define a custom connection string that encapsulates all the information required to connect to the data source, including the location of the database and security information needed to access the data. In the following example, you'll concentrate on using a System DSN.

There are no real advantages to be had in using one method over the other. It might simply come down to your hosts not allowing you to create a system DSN on their servers. If this were the case, you would then need to use a custom connection string, known as a DSN-less connection.

This can all be done from Dreamweaver, but it's just as easy to go out to your operating system and set it up from there—the dialogs that are used are exactly the same, but getting to them requires fewer clicks on your part (and far less explanation on mine).

Setting up a DSN to an Access database

1. Open the ODBC Data Source Administrator on your computer by going to Control Panel ➤ Administrative Tools ➤ Data Sources (ODBC).
2. Click the System DSN tab, and then click Add.
3. Select Microsoft Access Driver (*.mdb) from the list, and click Finish.
4. Now you get to enter the name of this DSN. Name it `DSNmysampledatabaseAccess` (or choose another name of your liking).
5. Leave the Description field empty unless you really want to enter something in there—it's not necessary for this to work.
6. Next, click Select to select the location of your database. Use the Select Database dialog to find the database and select it. Click OK, and the full path to your database will be shown above the Select button. Mine is `C:\databases\mysampledatabase.mdb`.
7. Click OK twice to close the two open dialog boxes.

Your DSN is set up and ready to use.

Setting up a DSN to a SQL Server database

1. Open the ODBC Data Source Administrator on your computer (Control Panel ➤ Admimistrative Tools ➤ Data Sources (ODBC)).
2. Click the System DSN tab, and then click Add.
3. Scroll the list to the bottom, select SQL Server and click Finish.
4. Name the data source `DSNmysampledatabaseSQL` (or choose another name of your liking).
5. Either type the name of your SQL Server into the Server drop-down list, or click to select it. Then click Next.

> *You'll need to specify the IUSR security account as the account that can log in to the SQL Server database using this connection. Earlier in this chapter, we covered setting up the IUSR\<machine_name> account—this is the account that you need to use here now. You should select SQL Server authentication and then enter those username and password details in the relevant fields in the dialog.*

6. If you've followed this chapter from the start, then the next few screens will be filled in for you. The database is already set to `mysampledatabase` because that's what you set it to when you created your login in SQL Server. The other settings you can leave in place. Click Next, then click Finish.
7. You now have the chance to test these settings to make sure no errors have crept in. Click Test Data Source and, all being well, you should see a screen that contains the words Tests Completed Successfully.
8. Click OK twice to get back to the original DSN screen, where you'll see your newly created DSN in the list.
9. Click OK to close this screen.

Connecting from Dreamweaver

The final stage for connecting your website to the database is to define a connection in Dreamweaver that will use the DSN that you just set up on your computer. This connection will be stored in a Connections folder in your site definition, and will be referenced on every page that you create that needs to talk to the database. The beauty of doing it this way is that if your connection details change for some reason, you'll only need to change one connection file to bring your entire site up to date, rather than having to modify every single page by hand.

To define your connection, open the Databases panel in the Application panel group. Click the plus (+) button and select Data Source Name (DSN) from the pop-up menu.

In the dialog that pops up, you're going to name this connection and select the DSN on your computer that it should use. You'll also need to provide the username and password that the DSN needs to authenticate itself with in the database, if required. In my Access examples, I didn't set a username or a password, so these can be left blank. In my SQL Server example, I had to specify the username but I left the password blank.

For Access databases, fill out the dialog as shown in Figure 4-21, and click OK.

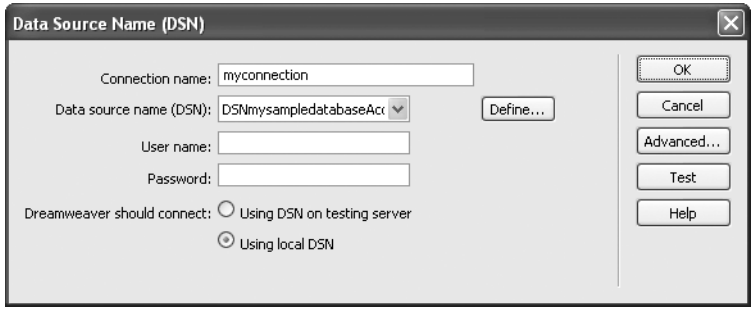


Figure 4-21. Creating a connection to Access in Dreamweaver

For SQL Server databases, fill out the dialog as shown in Figure 4-22, and click OK.

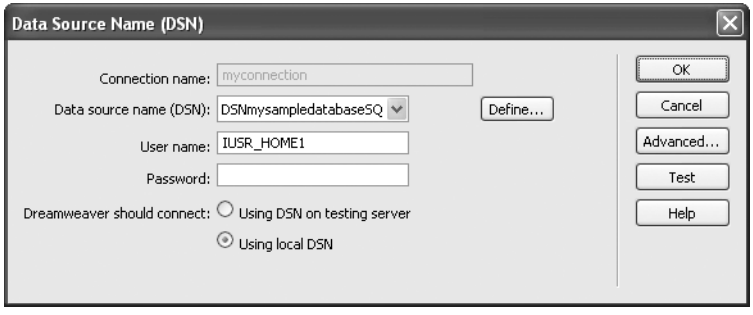


Figure 4-22. Creating a connection to SQL Server in Dreamweaver

Once your connection file has been created within your site, you can start to build recordsets on your web pages.

> *If you start to build a recordset before a connection file has been created for your site, then the* Connections *drop-down list in the Recordset dialog will be empty; however, you can create the connection by clicking the* Define *button next to the* Connections *drop-down list and following the steps outlined previously.*

The Simple Recordset builder

The Simple Recordset builder allows you to build an SQL statement to request data from your database. You won't always need to specify a `WHERE` clause in your SQL statement, but if you do, the Simple Recordset builder allows you to specify a single `WHERE` clause. If that's not enough for your purposes, then you'll need to use the Advanced Recordset builder (the Advanced Recordset builder will be covered in the next section).

There are only a few preparatory steps you need to follow to be able to use dynamic data on a web page. The first thing to do in Dreamweaver is set up a site definition in which you're going to build your web application. After you've created your site definition, you'll create a new page. You'll specify that this page is a dynamic page, and also which server-side scripting language you're going to use (ASP VBScript in your case). For further detail on the steps briefly mentioned here, see the "Defining an ASP VBScript site in Dreamweaver" section of Chapter 2.

Once you've created a page that you can add data to, you then need to create a recordset. Rather usefully, the Bindings panel found in the Application panel group tells you this, as shown in Figure 4-23.

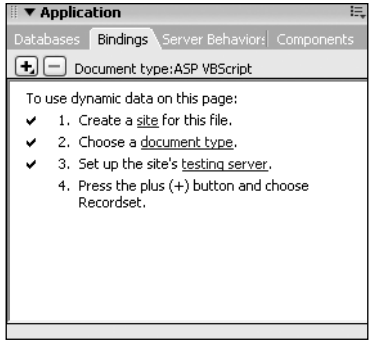

Application
Databases
Bindings
Server Behaviors
Components
Document type:ASP VBScript
To use dynamic data on this page:
1. Create a site for this file.
2. Choose a document type.
3. Set up the site's testing server.
4. Press the plus (+) button and choose Recordset.

Figure 4-23. The Bindings panel

To build a recordset using the Simple Recordset builder, click the plus (+) button in either the Bindings panel or the Server Behaviors panel, and select Recordset (Query) from the pop-up menu (the first option on the menu).

This will present you with the Simple Recordset builder interface, where you can set about defining what data you want to retrieve from the database. To retrieve the whole of the `viewQuotes` table, you would complete the dialog as shown in Figure 4-24.

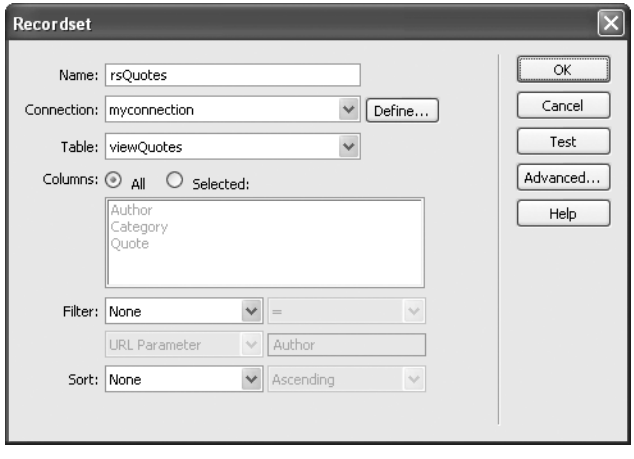


Figure 4-24. The Simple Recordset builder dialog box

The Simple Recordset builder also allows you to specify a single `WHERE` clause and a single `ORDER BY` clause.

To illustrate just how easy this is, let's specify that you want to apply a filter to the `Category` column, and specify that the category must equal `Computers`. You'll also set an `ORDER BY` clause to sort these records alphabetically by the person who authored them.

To do this, select Category from the Filter drop-down list and select the equality operator (=) from the drop-down list to the right of it. Then select Entered Value from the list below the Filter drop-down list, and in the text box to the right of that, type Computers. In the bottom-left drop-down list (labeled Sort), select Author; and in the bottom-right drop-down list, select Ascending. You should now have something that looks like Figure 4-25.

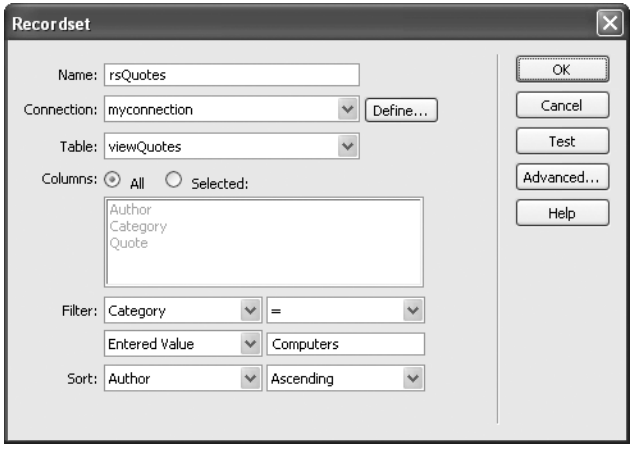


Figure 4-25. The completed Simple Recordset dialog

Once you've selected all the relevant settings, click OK, and the recordset will be created for you and added to the Bindings panel. If you then click the plus icon next to the recordset name, you'll see all the columns of the viewQuotes view, as shown in Figure 4-26. If you so desire, you could then start adding these pieces of data to your web page.

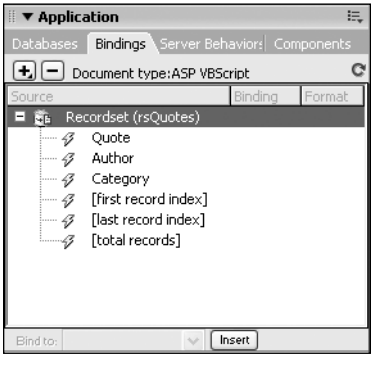


Figure 4-26. The recordset being displayed in the Bindings panel

With the recordset created, you can see that it not only retrieves all of the columns from the viewQuotes view, but it also adds three extra items: [first record index], [last record index], and [total records].

These three elements are useful for signposting data. For example, if you create a search results page that will display ten results per page, you could include these three elements on the page somewhere to indicate which section of records the user is currently looking at—something like Currently viewing 1 to 10 of 73 records. In this example, the 1 is produced from [first record index], the 10 is produced from [last record index], and the 73 is produced from [total records]. If you use a link to get the next page of records (next 10), then the first two signposts would change to Currently viewing 11 to 20 of 73 records.

To make this simple recordset a lot more flexible and far more useful for a web page, instead of specifying that the Category must be Computers, you can make it use a parameter that your web page must pass to this recordset to use as the filter.

Open up the Recordset dialog for this recordset again by double-clicking the recordset name in the Bindings panel. Where you previously selected Entered Value from the drop-down list, this time select URL Parameter. Where you previously entered Computers in the text box to the right of that drop-down list, type cat. This will be the name of the URL parameter that this recordset will use to filter database with (see Figure 4-27).

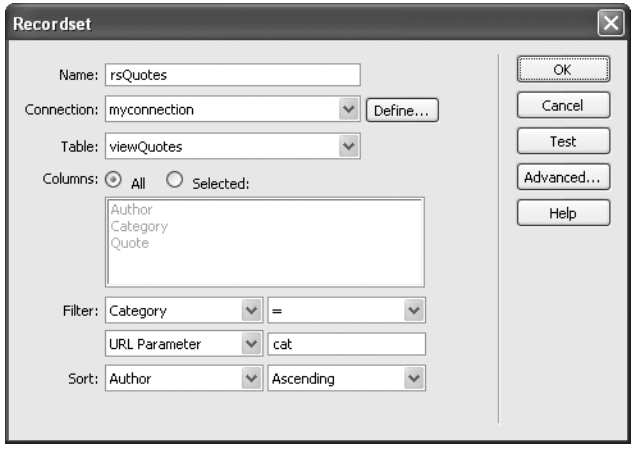


Figure 4-27. Specifying a URL parameter name in the Recordset dialog to apply a filter to the recordset

If this recordset were on an actual web page that displayed data, and you browsed to that page, specifying a URL parameter of `cat=computers`, the recordset would be filtered to show only quotes in the `Computers` category. If the URL parameter were `cat=technology` then only quotes from the `Technology` category would be shown. It's easy, flexible, and powerful—what more could you want?

The Advanced Recordset builder

The Advanced Recordset builder is the place to build those recordsets that require more detail than can be offered by the Simple Recordset builder in terms of their SQL statement. The Simple Recordset builder is only capable of creating recordsets that use either no criteria or one criterion.

Once you know how to work it, creating simple or advanced recordsets in this dialog is limited only by your knowledge of SQL. The basic layout of the SQL statement is put into place as you click through the building process. After that, you'll need to dive into the code and enter the final details. Don't worry; it's easier than it might sound.

In the next example, you'll build the same recordset as you did with the Simple Recordset builder, but this time you'll add an extra parameter into the mix to illustrate some of the power that the Advanced Recordset builder affords you.

You start the same way as before: click the plus (+) button in either the Bindings panel or the Server Behaviors panel, and select Recordset (Query) from the pop-up menu. On the right-hand side of the Simple Recordset builder dialog, there are five buttons—press the Advanced button to switch to the Advanced Recordset builder.

> *When you build a recordset and click* OK*, the dialog you used to build that recordset is the one you'll see the next time you go to build a recordset.*

Now you'll see the empty Advanced Recordset builder with which you're going to build your SQL statement.

The first thing you need to do is name the recordset—you can call it `rsQuotes`, as in the last example, if you like. You can't have two recordsets with the same name on the same page in Dreamweaver, so if you already have an `rsQuotes` recordset, you can delete it now for your testing purposes.

Now you need to select the database connection so that you can view the database objects in the Database items window at the bottom of this dialog. If you click the plus (+) icons next to the three items in there (Tables, Views, and Stored Procedures) before selecting your connection, they won't contain anything. Once you've selected your connection, these items will be populated with your database objects, and you can then go ahead and build your SQL statement.

Select your connection from the Connections drop-down list. In the Database items window of this dialog, click the plus (+) icon next to Tables, and a listing of all the tables in your database will be displayed (there might be a short delay while the connection is made to the database). Click the plus (+) icon next to viewQuotes, and the list will be updated to show all the columns of the `viewQuotes` view.

Ordinarily, you would click the table name to highlight it and then click the SELECT button to the right of the Database items window to start creating your SQL statement. However, there's a little shortcut here: you want to select all the columns for all the records in this table that have a value in the `ProductCategory` column that matches your criteria. The quick way to generate the SQL statement for this is to click the Author column and then click the WHERE button to the right of the Database items window.

The SQL window will now show the beginnings of your SQL statement, as shown in Figure 4-28.

You now have an incomplete `WHERE` statement that, if left as it stands, would make this recordset fail. You need to define what that `Author` column should equal.

Click to place the cursor after Author in the SQL window and type = 'varAuthor'.

You have total control over naming your variables—I choose to prefix mine with `var` and give them meaningful names, which lets me know that they're variables and gives me a good idea about what their contents are.

You know this variable is going to contain a text-based value, so it must be wrapped in single quotes to identify it as such.

Next, you need to define what value `varAuthor` should pass to this SQL statement. You do this using the Variables window, which is located below the SQL window. Click the plus (+) icon above the Variables window, and a new variable line will be added to the Variables window. Next, click the Name column of this new variable line and type varAuthor. Click in the Default Value column (or press Tab) and type: xyz. Then click in the Run-Time Value column (or press Tab again) and type: Request("author").

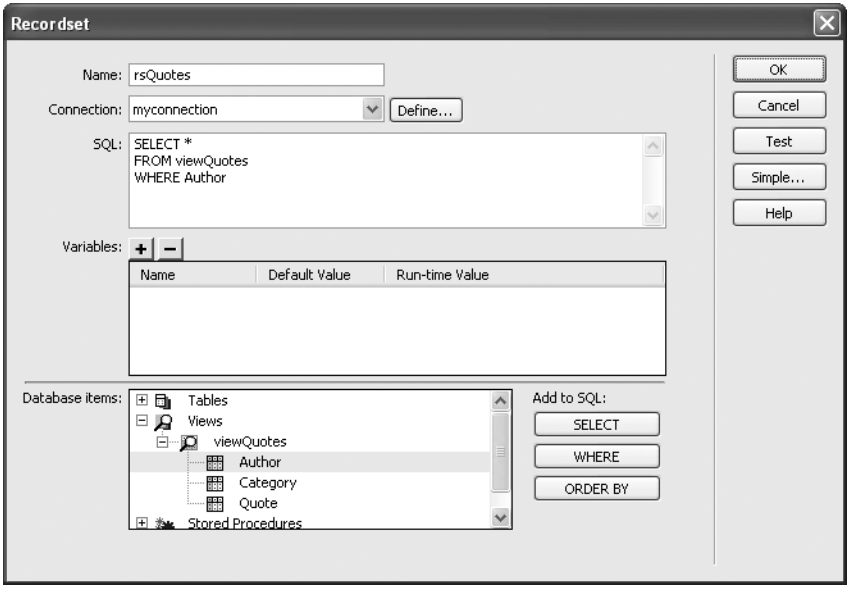


Figure 4-28. The SQL statement being built in the Advanced Recordset builder dialog

To explain what's going on here—in your SQL statement in the SQL window, you've said that the `Author` column should equal the value contained in the variable called `varAuthor`. The default value you supplied serves two purposes: first, it allows you to test your completed SQL statement to make sure that you're getting the results that you expect back; and second, when the page is used in a live environment, it acts as the actual variable value if the URL parameter called `cat` doesn't exist.

The Run-Time Value column is where you specify what should be used as the actual value to filter this recordset. As previously mentioned, if this parameter doesn't exist, then the default value will be used to filter the recordset. I use a default value of xyz to ensure that the recordset will be empty if the `cat` parameter doesn't exist. You may want to use a default value of an existing `Author` name to ensure that this recordset always contains data.

Next, click Category to select that column and click WHERE to add it to the SQL `WHERE` clause. You'll be creating a recordset that's filtered by `Author` and `Category`—something that the Simple Recordset builder can't do.

Click to place the cursor after Category in the SQL window and type = 'varCategory'.

Now you need to define what value `varCategory` should pass to this SQL statement. In the Variables window below the SQL window, click the plus (+) icon again to add another line to the Variables window. Then click in the Name column of this new variable line and type varCategory; click in the Default Value column (or press Tab) and type xyz; and click in the Run-Time Value column (or press Tab again) and type Request("category").

Most of the recordset is now built. Remember that in the Simple Recordset example, you added a sort order to your records that would return the records ordered by the author's name. To accomplish this

in the Advanced Recordset builder, select Author in the Database items window and click the ORDER BY button to the right. As previously discussed, ORDER BY defaults to *ascending* order, so you don't need to specify that unless you're adding more than one ORDER BY clause.

Your finished Advanced Recordset dialog should look like Figure 4-29.

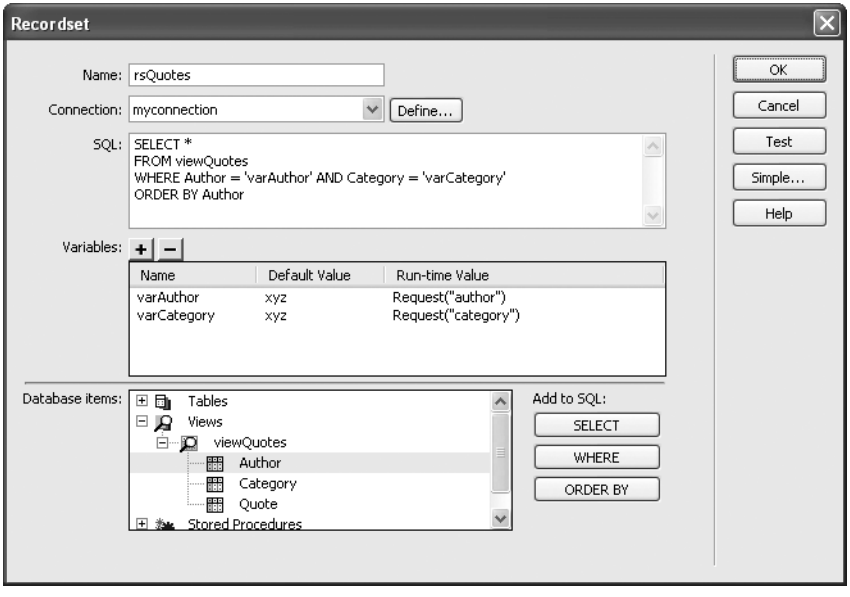


Figure 4-29. The completed Advanced Recordset dialog with two parameters added

Click OK to close the dialog and create the recordset.

Hopefully, you can see the advantages of using the Advanced Recordset builder over the Simple Recordset builder—you can use multiple parameters in your statements, and you can also just write your SQL statement directly in the SQL window by hand if you so desire.

Using commands

At some point in your website development, you'll no doubt want to add, edit, or delete data from your database. In Dreamweaver, there are two ways you can perform any of these actions. The first, and perhaps most common, is to use the standard Insert Record, Update Record, and Delete Record server behaviors, which are covered in Chapter 6.

The other way is to use a **command**. Commands can be created to perform all of the same database operations as their server behavior counterparts, with the addition of being able to utilize **stored procedures**. So, why might you choose to use commands over server behaviors?

If you're comfortable writing syntactically correct SQL statements by hand, or even if you just have an understanding of the correct formation of SQL, then you'll find the Command interface a breeze. It's very much like the Recordset builder interface.

If you need to have more than one of the same database interactions on the same page, then you'll need to use one command per database interaction. For instance, if you want to insert a new record into your database that needs to be split over two tables, you could set up two commands on a page to do this for you. The server behavior versions of these database interactions can only be included on your web page once.

Perhaps the major benefit of commands over server behaviors is the amount of code produced. Server behaviors tend to produce bulky code because they need to cater to a lot of different scenarios, whereas their command counterparts produce a small amount of code while maintaining adaptability.

The following examples will introduce you to the usage of each of the commands available to you, with the exception of the stored procedure command, which is beyond the scope of this book.

The creation process of each of these commands begins in exactly the same way. Click the plus (+) button in either the Bindings panel or the Server Behaviors panel and select Command from the pop-up menu. You'll see the Command dialog, which looks like Figure 4-30.

Your command will only be listed in, and therefore editable from, the Server Behaviors *panel after creation, no matter which panel you choose to start the creation process with.*

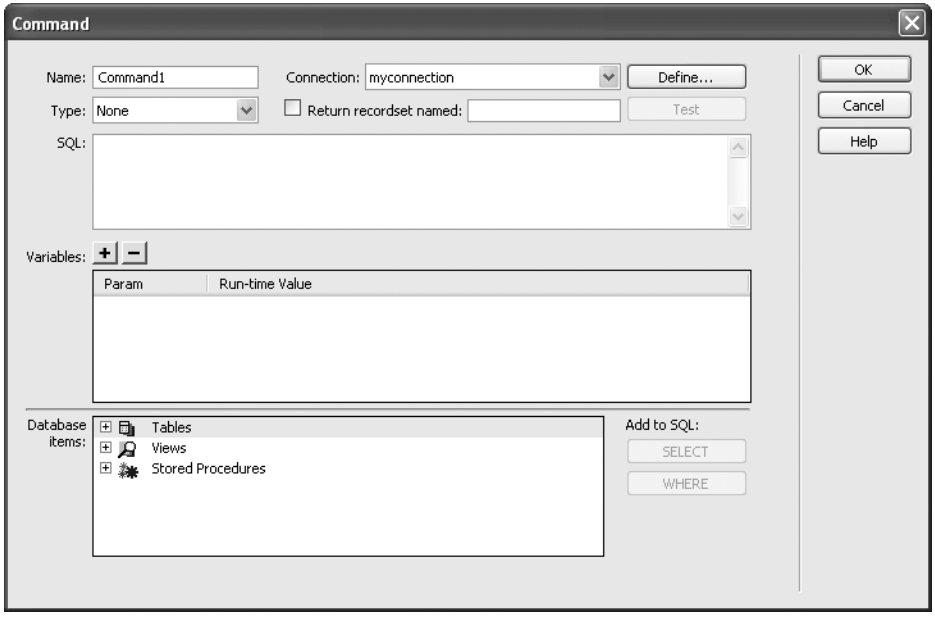


Figure 4-30. The Command dialog

The first thing to do in creating any command is to name it, which you can do in the top-left box of the dialog. All commands are given a default name, but it's not too descriptive, so you should always change that to something that makes more sense.

Inserting a record

The following example explains how to insert a record using a command. You'll be adding a new quote to the database, so you might name your command `cmdInsertQuote`.

My connection is already selected for me because I've already set up a connection for this site. If yours isn't, then you'll need to set up a connection before continuing, which you can do by clicking the Define button in this dialog (or follow the steps outlined earlier in this chapter in the "Making the connection" section).

From the Type drop-down list, you want to select Insert because this is an `INSERT` command. (Easy, isn't it?) Immediately after you select Insert from the list, the basic `INSERT` statement is added to the SQL window. All you're going to do is populate this statement with the relevant information. You can ignore the check box next to Return recordset named, as this is used only for stored procedures.

The buttons in the lower-right of the dialog, under the Add to SQL heading, change to reflect the type of statement you're building. For the `INSERT` statement, only one of the two buttons is active—it's called COLUMN.

In this case, you want to insert only certain pieces of information into the `tblQuotes` table. In all likelihood, those pieces of information would have been submitted from a form on the previous page. With this in mind, you need to select the columns you're inserting your data into from `tblQuotes` in the Database items window, and then click the COLUMN button.

Select AuthorID and click COLUMN. The SQL window is updated to reflect your selection. Then, select CategoryID and click COLUMN. Finally, select Quote and click COLUMN again. The SQL window should now read as follows:

```
INSERT INTO tblQuotes (AuthorID, CategoryID, Quote)
VALUES ( )
```

All you need to do is tell the command what values to insert into the specified columns. An important point to note is that you can't mix up the items between lists; they must be in the exact same order. In other words, the order in which you specify the items in the `INSERT INTO` line must be the same order in which you specify the items in the `VALUES` line.

To get values into this `INSERT` command from a fictitious form on the previous page, you need to request that information and use variable placeholders in the values listing, as follows.

Click inside the parentheses after `VALUES`, and enter the following string of variable names, each separated by a comma (don't forget, text-based values need to be surrounded by single quotes):

```
varAuthorID,varCategoryID,'varQuote'
```

Now you need to define where these variables are going to get their information. This example still assumes that you're getting your information from a fictitious form on the previous page and that the form elements were named as shown in the Request parts of the following details.

Click the plus (+) button above the Variables window, and then click in the Name column and type varAuthorID. Tab to the Run-Time Value column and type Request("AuthorID"). Add another variable line for varCategoryID and enter a runtime value of Request("CategoryID"). Finally, repeat the process once more, this time with a name of varQuote and a runtime value of Request("Quote").

Your Command dialog should now look like Figure 4-31.

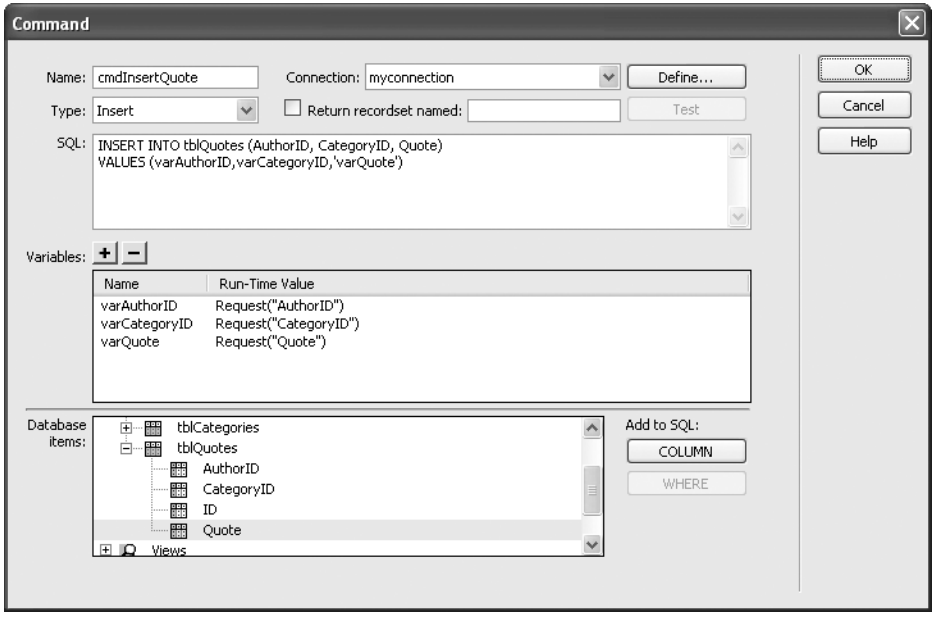


Figure 4-31. The completed Command dialog, ready for data to be inserted

With all that done, click OK, and your command will be applied to the page.

Now, when the form on the previous page submits its information to this page, it will be inserted into the database.

Updating a record

This UPDATE command will update the AuthorID of a record (for instance, the one you just inserted with the INSERT command).

With the Command dialog open, name this command cmdUpdateAuthorID, and select Update from the Type drop-down list. Your connection should already be selected in the Connection drop-down list. The two buttons in the Add to SQL area have the labels SET and WHERE, because that's the syntax of the UPDATE statement, as you'll see shortly.

In the Database items window, expand the Tables listing, and then expand the tblQuotes table. Click to highlight the AuthorID column, and then click the SET button. Your SQL window will be updated to show the following SQL code (this example uses Access—the SQL Server version of this would be the same but with dbo. prefix on the table name):

```
UPDATE tblQuotes
SET AuthorID
WHERE
```

Now you need to tell the command what value AuthorID should be set to, and you also need to identify which record this statement should be applied to. You do this by typing = 'varAuthorID' after AuthorID. The SET line of the SQL window will now read as follows:

```
SET AuthorID = varAuthorID
```

varAuthorID, as you've probably guessed by now, is a variable that will get its runtime value from a field called AuthorID, which you'll assume has been submitted from the previous page. Click the plus (+) button above the Variables window and add varAuthorID as the variable name and Request("AuthorID") as the runtime value. The only thing left to do is to make sure that you're going to update the right record in your database.

You do this by using the ID number of the record, which will also have been passed through to this page from the previous page. Click ID in the Database items window and then click WHERE. Click in the SQL window after ID and add = varQuoteID. You now need to add that variable to the variables list, so click the plus (+) button above the Variables window again, and on the new variable line, set the variable name to varQuoteID and the Run-Time Value to Request("QuoteID").

Your finished Command dialog should look like Figure 4-32.

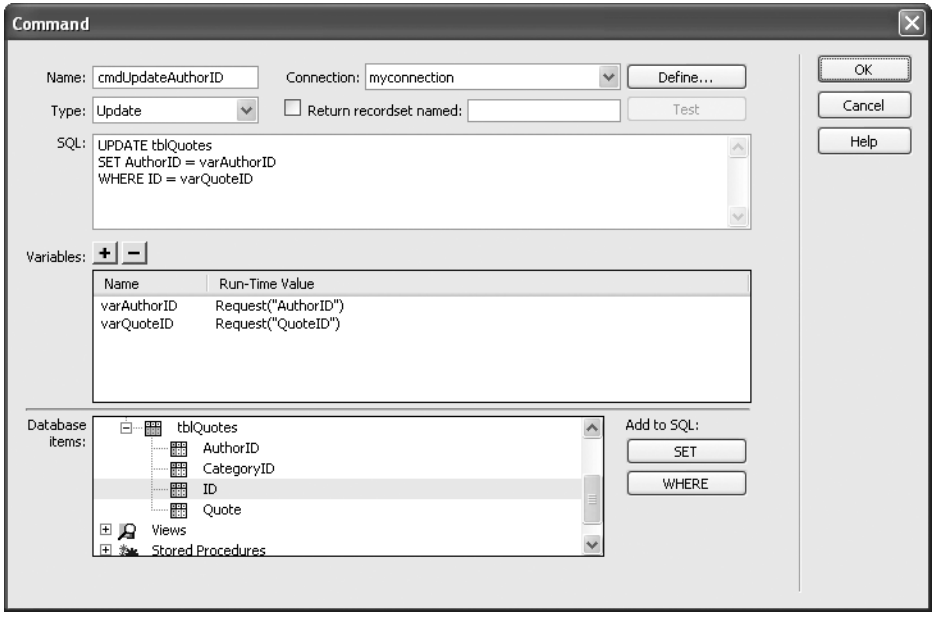


Figure 4-32. The completed Command dialog, ready for data to be updated

Updating multiple records (a simple example)

You'll often want to be able to update more than one user at a time, so that, for example, you can activate or deactivate several accounts at once. In the following example, suppose you have a page that displays all of your website administrators, and against each one is a check box that can be checked and unchecked to activate and deactivate that administrator.

With the Command dialog open, name this command `cmdUpdateMultiAdmin` and select Update from the Type drop-down list (your connection should already be selected in the Connection drop-down list). Expand the Tables listing in the Database items window, expand the `tblAdmin` table, highlight `Included`, and click SET.

Your SQL window should now display the following code:

```
UPDATE tblAdmin
SET Included
WHERE
```

Now type = 0 after `Included`. This column of the database is a binary column, so a 1 (SQL Server) or -1 (Access) in this column would mean that the record is included—a 0 means it is excluded.

Now you need to specify which records to update. You'll be using the `IN` keyword in this case (see the "Useful SQL Keywords" section earlier in the chapter for more information on the `IN` keyword). In the Database items window, click ID and then click WHERE. `ID` will be added to the SQL window. Click in the SQL window to place the cursor after `ID`, and type the following:

```
IN (varID)
```

Your SQL should now look like the following:

```
UPDATE tblAdmin
SET Included = 0
WHERE ID IN (varID)
```

All you need to do now is add `varID` to the Variables window. Click the plus (+) button above the Variables window and enter the variable name (`varID`) in the Name column. In the Run-Time Value column, type Request("ID").

The finished multiple update Command dialog should look like Figure 4-33.

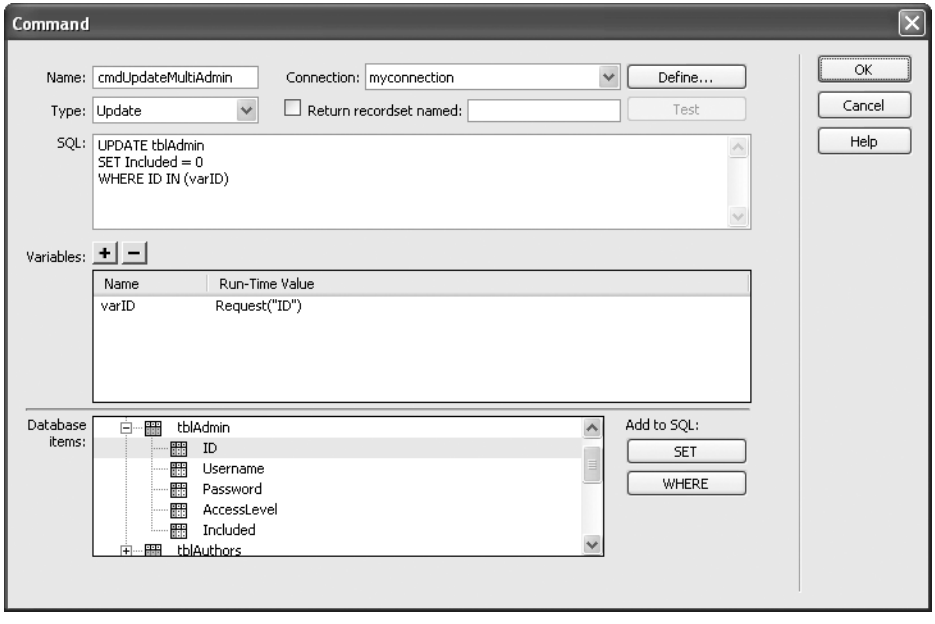


Figure 4-33. The completed Command dialog, ready for multiple rows of data to be updated

Now, when multiple check boxes are ticked on the page that submits to this page, a list of IDs will be delivered in a comma-delimited string, and all the records that match one of those IDs will have their `Included` column updated to contain a `0`. When this happens, these administrators will no longer be able log in—as long as the correct logic is in place to assure that the `Included` column is used, of course!

Deleting a record

Deleting records is sometimes a necessary step for keeping your database up to date. The following example shows you how passing the ID number of a product record to a `DELETE` command will delete that product.

With the Command dialog open, give the command a name of `cmdDeleteRecord` and select Delete from the Type drop-down list. Your connection should already be selected in the Connection drop-down list. The shortcut detailed in other commands doesn't work for the `DELETE` command, you have to specifically click the table name in the Database Objects window and click the DELETE button to add the table name to the SQL.

Then, expand the `tblQuotes` table in the Database Objects window and click the ID column. Click WHERE to add this column to the SQL window.

Your SQL window should look like this:

```
DELETE FROM tblQuotes
WHERE ID
```

You now need to place the cursor after ID and type = varID**.**

Add the variable to the Variables window by clicking the plus (+) button above it. In the Name column, type varID and set the Run-Time Value to Request("ID").

Your final Command dialog should look like Figure 4-34.

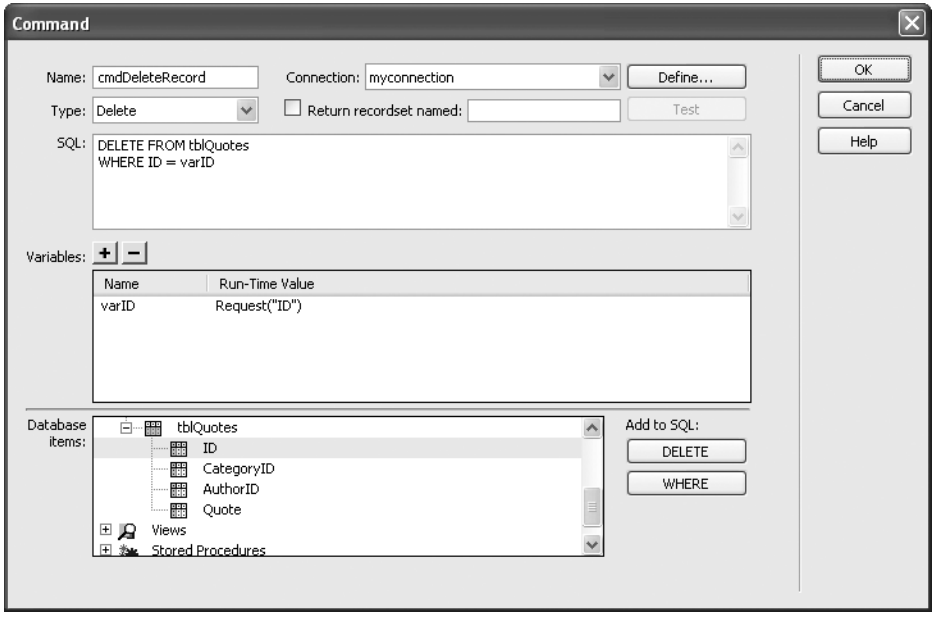


Figure 4-34. The completed Command dialog, ready for data to be deleted

Click OK to apply this command to the page. When this page is passed the ID number of a record in the ProductID parameter, that record will be deleted.

Deleting multiple records (simple example)

Following on from the previous example, a common scenario would be the need to delete multiple items from your database. You could do this one at a time if you want, but if you need to delete lots of items then I wouldn't recommend it. What I would recommend, however, is using a command to delete multiple items, as shown in the following example.

With the Command dialog open, give the command a name of cmdDeleteMultiQuotes and select Delete from the Type drop-down list. Your connection should already be selected in the Connection drop-down list. In the Database items window, select the table that you're deleting the products from—in this case, tblQuotes—and click the DELETE button. Then click the ID column and click the WHERE button.

Your SQL window should now look like the following:

```
DELETE FROM tblQuotes
WHERE ID
```

Now place the cursor after D and type: IN (varID).

Click the plus (+) button above the Variables window to add a new variable line. Click in the Name column and type varID. Tab to the Run-Time Value column and type Request("ID").

Your final Command dialog should look like Figure 4-35.

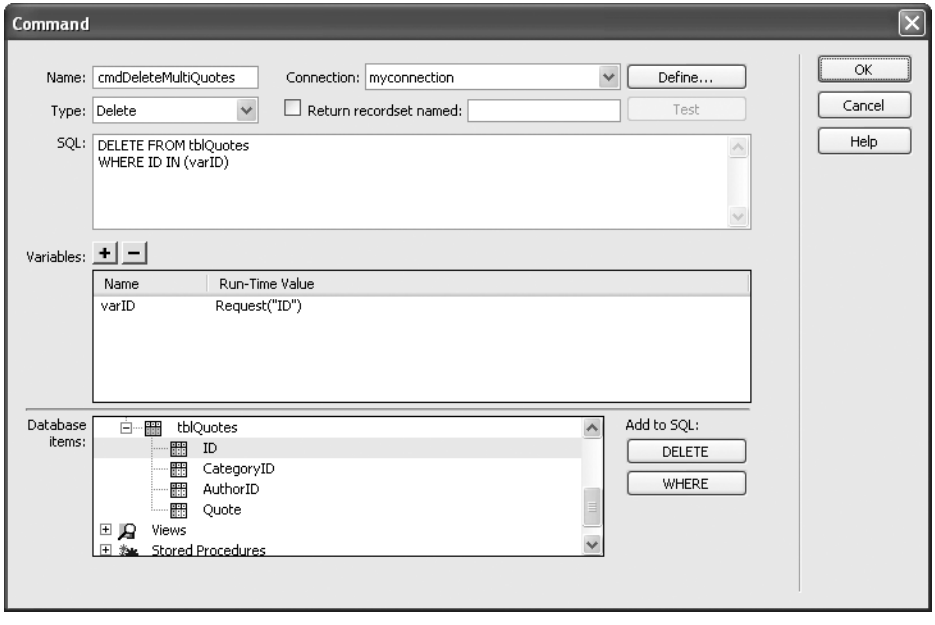


Figure 4-35. The completed Command dialog, ready for multiple rows of data to be deleted

Click OK to apply this command to the page. When this page is passed the ID number of one or more records in the ID parameter, those records will be deleted. You can pass a comma-delimited string of numbers into this command, and it will delete each record whose ID is contained in that string.

Conclusion

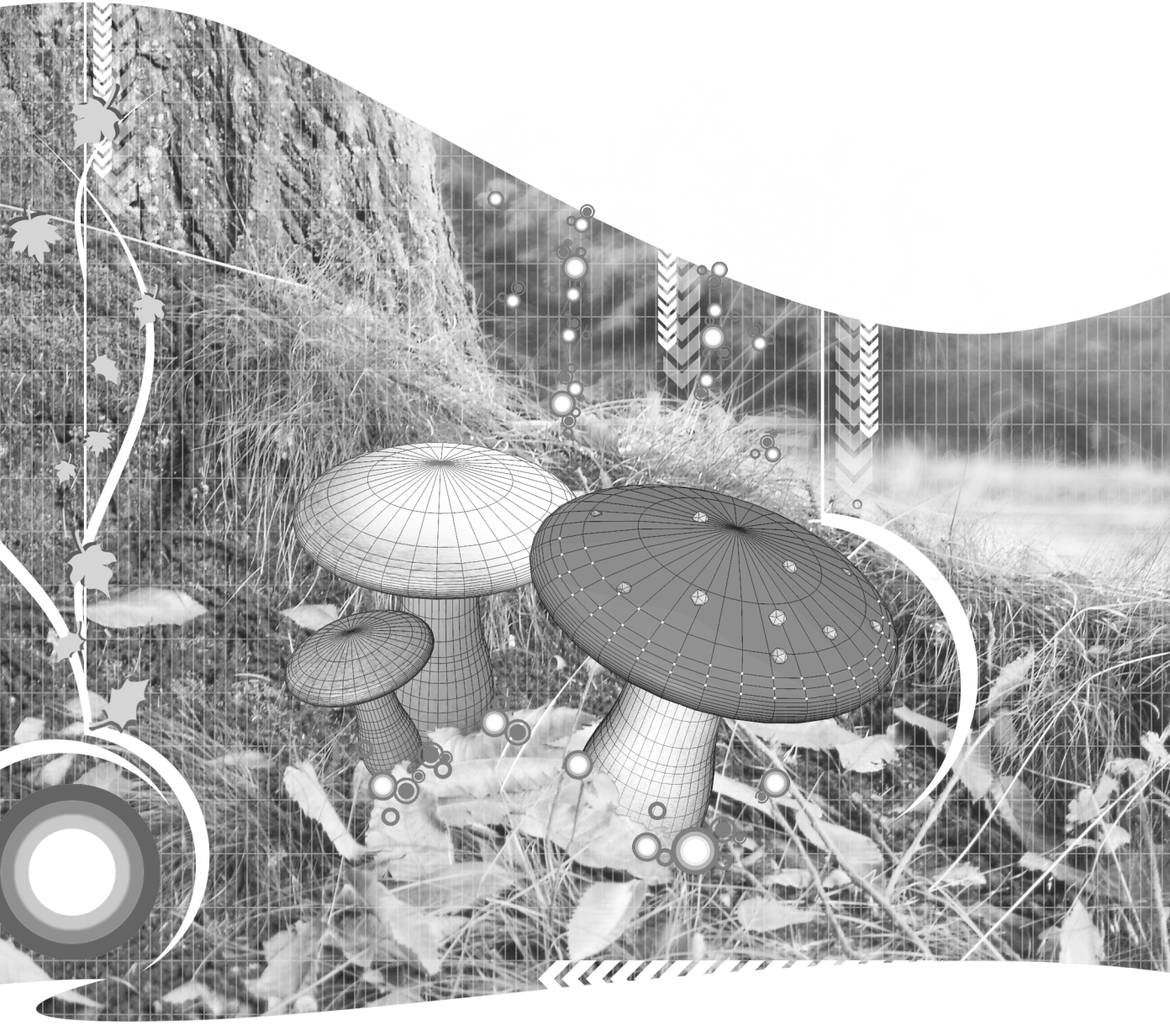
This chapter has given you insight into the basics of the SQL language and has shown you a few pointers on database design pitfalls to avoid. It has covered the database creation process in Access and SQL Server, along with how to create tables and views inside a database and build relationships between objects to enable referential integrity on your data.

This chapter has covered some fundamental SQL and provided a good introduction to using SQL in your data-driven applications, along with about a basis in some of the more common keywords of everyday SQL.

It has covered how to get connected to the database from within Dreamweaver, and it has also covered the command functionality that Dreamweaver presents you with. Hopefully, I've prompted a few ideas of the possibilities available to you when working with commands by detailing these few simple examples and explaining their usage.

When it comes to using the power afforded you by SQL, you've only really scratched the surface here, and I heartily recommend that you seek out a good book solely dedicated to the subject of SQL to take your knowledge on to the next level.

With careful planning and good database design, the knowledge you've gained from this chapter can take you a long way in your web application development projects.



Chapter 5

WORKING WITH FORMS

Forms allow you to receive dynamic data from and send data to your website's users. Forms are the foundation for web applications that collect data, and they are one of the most important methods for gathering information from your web users onto your server. They are also the basis of how users do things such as registering with, logging in to, and sending feedback to your website.

During the "static" HTML days, developers had no idea how to make real use of forms, or they tried to find some CGI script to do the work of forms for them. The good news is that with our newfound knowledge of ASP and VBScript, forms are now far more useful than they ever were in the past!

This chapter covers how to exchange data with your users through online forms. You'll learn how to enable your users to submit data through an online form and have the results displayed, sent to an e-mail, and entered into a database. You'll then move on to see some practical ASP examples that allow users to actually interact with and submit data to your web applications. Here are some of the key topics we'll cover:

- Understanding the `get` and `post` form methods
- Retrieving and displaying values entered in various form elements
- Making web applications respond differently according to form values entered
- Sending and receiving URL parameters, otherwise known as querystrings, throughout web pages

- Setting up a Simple Mail Transfer Protocol (SMTP) server locally and connecting to a remote SMTP server
- Sending form values to an e-mail using several mail components
- Sending web users' forgotten passwords to their e-mail addresses
- Sending e-mails to a list of e-mail addresses stored in a database

> *Since you will be completing a few examples during the course of this chapter, we recommend creating a subfolder titled* form_examples *within the root of your site to stay organized.*

Form attributes: Action and Method

Before you dive into creating that first dynamic form, you will need to know about a couple of important attributes. The most important attributes for the HTML form tag, `<form>`, you need to be concerned with for this chapter are `Action` and `Method`.

For the `Action` attribute, you must specify the page on the server that will actually process the form and its elements, as you will see later in this chapter. When you highlight the form in Dreamweaver, you will be able to browse your computer for the appropriate page from the Properties panel next to the heading Action. You can place the code that will process the form on the same page that contains the form, or you can put the code on another page. In either case, the `Action` attribute needs to know which page will perform the processing on it.

The `Method` attribute has only two possible values. The two types of form methods are `get` and `post`. The following sections explain each method.

get method

When a form is submitted using the `get` method, the form field values will automatically be appended to the URL, in name/value pairs called **URL parameters**. Each pair is separated by an ampersand (&). For example, say you enter the text hello in a form text field named `greeting` and submit the form to a page named `get_form.asp`. Once the page is submitted, you will see the form field value in the address bar, for example: `http://localhost/webprodmx/get_form.asp?greeting=hello`.

> *The part of the URL after the question mark (?) containing the URL parameters is called the* ***querystring****.*

Using the `get` method can make troubleshooting a little simpler because the values are will be obvious in the address bar; you will know which values have been passed and which haven't, whereas when they are passed using the `post` method, you won't be able to see the form values on the next page

unless you explicitly print them out. However, be aware that the client can also see these values, and the URLs are cached by the browser (this means that you should not use this method when submitting sensitive information such as credit card numbers). Also, the size of the values passed is restricted to the maximum length of characters allowed in the address bar. Nonetheless, you will learn how to retrieve the values of URL parameters later in this chapter.

post method

Using the post method, the form field values are submitted in the HTTP header. This is the preferred method because the data submitted is not directly visible to the client. Another advantage is that you can send much larger pieces of information using post than you can with get.

> *You will be using the* post *method regularly in the exercises in this book. Make sure to specify* post *when you create your forms in Dreamweaver. If you need to troubleshoot the form using the* post *method, simply print out the form values on the subsequent page, which you will learn how to do in the next section.*

Retrieving form values with Request.Form

Request.Form retrieves any form element submitted with a post method. Whenever you create a form element, you give it a name. For example, <input type="text" name="username"> will create a text field named username. In order to display the value that was inputted in this text field and submitted, you need to use Request.Form("username") on the page that received the submission. That username is contingent upon the name of the text field—if you change the name of that text field, then you must change the name in the ASP code (i.e., they must match each other). Also remember that ASP code always needs to be within ASP delimiters. If you want to print a form field value to the screen, use the following code:

```
<%=Request.Form("username") %>
```

> *Always make sure that the name of the form element and the value of the intended* Request.Form *match.*

Creating sample login forms

Now let's look at some code in action. In this section, you'll create various form elements with the post method: text boxes, list/menu elements, and check boxes. You'll also learn how to retrieve data via the get method and a URL parameter.

Forms with text box elements

In this example, you'll create a sample login form and display the username on the next page using **text boxes**. Start by creating a form in Dreamweaver with the following properties, and save the page as `login_user.asp` (see Figure 5-1):

```
<!-- login_user.asp -->

<form name="frm_login" method="post" action="welcome_user.asp">

  username: <input type="text" name="username" /><br />
  password: <input type="text" name="password" /><br />
  <input type="submit" name="submit" value="submit" />

</form>
```

Figure 5-1. Login form displayed in browser

Now create a second ASP page named `welcome_user.asp` to retrieve values submitted through this form:

```
<!-- welcome_user.asp -->

Welcome <strong><%=Request.Form("username")%></strong>!<br />
I know your password too but I won't display it ;)
```

If you access `login_user.asp` through your web server and enter your username as Omar, you will see the result shown in Figure 5-2 on the `welcome_user.asp` page after clicking Submit.

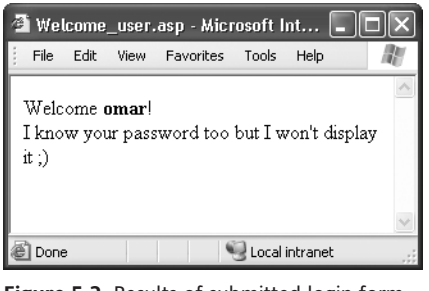

Welcome_user.asp - Microsoft Int...
File Edit View Favorites Tools Help
Welcome omar!
I know your password too but I won't display it ;)
Done
Local intranet

Figure 5-2. Results of submitted login form

Notice how the code <%=Request.Form("username")%> in welcome_user.asp retrieves the value of the <input type="text" name="username"> form element from the login_user.asp page. Certainly you can retrieve the other form elements on this page ("password" and "submit") using Request.Form("password"), and Request.Form("submit"). In this example, we chose not to display the password because in most real-world login applications, the password is merely used for authentication purposes and is rarely displayed to the page, especially for security purposes.

> Request.Form *will only retrieve values of form elements that were directly submitted to it. In other words, you cannot display the form element values on a third page using* Request.Form *unless you passed the values from the second to the third page with another form (or through another means of transferring values, such as querystrings, sessions, cookies, etc.) that had its default values the same as the first form.*

The key here is that you now know how to retrieve values submitted through forms. You can then do whatever you want with the value. Later in this chapter, for example, you will learn how to send these form values to an e-mail address.

Forms with list/menu elements

In this section, you'll learn how to retrieve and display the value selected from a **list/menu element**. When creating a form in Dreamweaver and adding a list/menu element, be sure to select the Menu type from the Properties panel. To start, in Dreamweaver create a new form with the following properties and save it as sex_selection.asp (Figure 5-3):

```
<!-- sex_selection.asp – allows for selection -->

<form name="frm_gender" method="post" action="display_sex.asp">

  <select name="sex">
    <option>male</option>
    <option>female</option>
  </select>
  <input type="submit" name="submit" value="submit" />

</form>
```

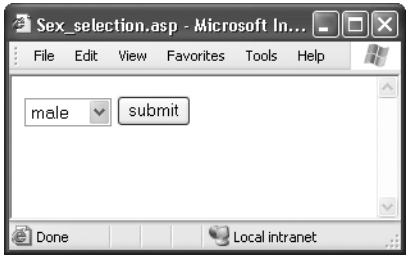


Figure 5-3. Form with menu element displayed in browser

Note that for menu elements, you can differentiate the values from the item labels that are displayed. You can access these options by highlighting the menu element and clicking the List Values button in the Properties panel. In the List Values dialog, you can change the values for each item (Figure 5-4). Note that the value will be retrieved on the subsequent page, not the label.

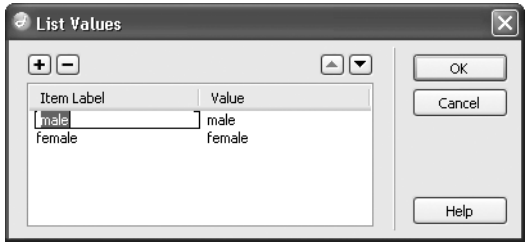


Figure 5-4. Specifying the URL parameters to be passed

Next, create a second ASP page with the following properties to retrieve the values of this element:

```
<!-- display_sex.asp – displays results -->

You are a <strong><%=Request.Form("sex")%></strong>
```

The results of one option are shown in Figure 5-5. To verify the results of the other option, go back and change your selection.

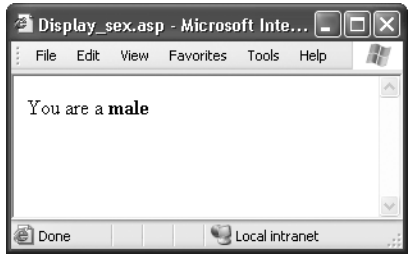


Figure 5-5. Sex selection form results

Retrieving the values of menu elements is done in the same way as retrieving values of text fields.

Menu element selection form with conditional logic

Using our knowledge of conditional logic, we can implement some tricks depending on the menu option the user selects. For example, we can change the code for `display_sex.asp` to redirect to different pages depending on the user's sex. Let's change that page.

```
<!-- display_sex.asp -- redirect based on gender -->

<%
If Request.Form("sex") = "male" Then
    Response.Redirect "http://www.menshealth.com"
```

```
  ElseIf Request.Form("sex") = "female" Then
    Response.Redirect "http://www.ivillage.com"
  End If
%>
```

The preceding code redirects the user to `http://www.menshealth.com` if the sex selection is male and `http://www.ivillage.com` if the sex selection is female. Of course, you could send males to `http://www.ivillage.com` and females to `http://www.menshealth.com` if you wish—the focus here is on making your code dynamic and responsive to an individual web user.

Since there are only two options, you could also use the following code:

```
<%
If Request.Form("sex") = "male" Then
    Response.Redirect "http://www.menshealth.com"
  Else
    Response.Redirect "http://www.ivillage.com"
  End If
%>
```

Instead of checking for both options, we check only for one. Since there are just two options, we know that if the response doesn't equal the option checked for in the code, it must equal the other option. You will likely find other uses for these tactics in your own web development.

Forms with check box elements

Check boxes are also very useful form elements. Each check box that is in a checked state submits a value when the Submit button is clicked. This can come in handy when you want a user to make a yes or no selection. To demonstrate, create a form with the following properties (Figure 5-6):

```
<!-- send_note.asp -- checkbox specification -->

Send us a note!

<form name="form1" method="post" action="send_note1.asp">

  name: <input type="text" name="name" /><br />
  email: <input type="text" name="email" /><br />
  message: <br><textarea name="message" rows=2 cols=20 wrap=hard>
</textarea><br />
  register for newsletter: <input type="checkbox" name="newsletter"
➥ value="1" />
  <br />
  <input type="submit" name="submit" value="submit">

</form>
```

Figure 5-6. Form with a check box specification

Now create the following ASP page to display the results:

```
<!-- send_note1.asp -->

Thank you <strong><%=Request.Form("name")%></strong>.<br />
You submitted the following message: <br />
<strong><%=Request.Form("message")%></strong><p>
Your email address is <strong><%=Request.Form("email")%></strong>.<br/>
<% If Request.Form("newsletter") = 1 Then %>

You also signed up for our newsletter.

<% End If %>
```

When `send_note.asp` is submitted (see Figure 5-7), `send_note1.asp` displays the results. In addition, if the user does select the check box the value, `1` will be submitted, associated with its name `newsletter`, and the `send_note1.asp` page will display the following (see Figure 5-8):

You also signed up for our newsletter.

Figure 5-7. Check box form with values entered

Figure 5-8. Results of the check box form

In the preceding few sections, you learned how to retrieve form elements submitted in a form via the `post` method in an ASP page. In the next section, you'll learn how to submit form elements using the `get` method and retrieve them from a URL parameter.

Receiving data from a URL parameter

Another way you can receive data from users is from a URL parameter. This method can be used (and very often is used) to pass data between pages. You can either allow users to pass data or you can embed your own data to be passed in the source code via an `href` attribute.

Before we delve into the details, you may want to know what a URL parameter is. A **URL parameter** or **querystring** is any attribute and value passed after a question mark (**?**) attached to the end of a URL. Here's an example:

```
http://www.website.com/webpage.asp?name=omar
```

The URL parameter in the preceding URL is `name=omar`.

Every URL parameter consists of a name and a value, just like any variable. The **name** of the URL parameter in the preceding example is `name`, and the **value** of the URL parameter called `name` is `omar`. If the URL parameter was `id=32`, then the name would be `id` and the value would be `32`.

You can pass multiple parameters by separating them with an ampersand (**&**), for example:

```
http://www.website.com/webpage.asp?name=omar&id=23
```

Retrieving URL parameters with Request.QueryString

So how do you retrieve these URL parameters? It's simple, and the procedure is exactly like `Request.Form`, but you use `Request.QueryString` instead. So, for example, if the URL parameter `http://www.website.com/webpage.asp?id=23` is passed, we can display it using `Request.QueryString("id")`.

To demonstrate, create the following pages:

```
<!-- display_users.asp -->

<a href="get_id.asp?id=12">Omar</a><br />
<a href="get_id.asp?id=15">Mona</a><br />
<a href="get_id.asp?id=145">Michael</a><br />
<a href="get_id.asp?id=148">Olivia</a><br />

<!-- get_id.asp -->

Look at the URL Parameter in the address bar<br>
The ID you selected equals <b><%=Request.QueryString("id")%></b>.
```

When you click one of the links in `display_users.asp`, the ID value contained in the corresponding URL is retrieved and displayed by `get_id.asp`.

You may recall from the earlier `get` and `post` discussion that using the `get` method when submitting a form appends all the form fields as name/value pairs in a query string of the action page (i.e., they are sent as URL parameters). The name sent will be the name of the form field, and its value will be the value entered or set for that particular form field. For example, if a user submits a form with a text field named `name`, clicks a Submit button named `submit` that has a value set to "send," enters omar as the name, and submits to a page called `thanks.asp`, the URL will look like this after submission:

```
thanks.asp?name=omar&submit=send
```

These values cannot be retrieved with `Request.Form`; rather, they are retrieved like normal URL parameters with `Request.QueryString`. Create the following pages to see an example:

```
<!-- Thankyou_note.asp -->

Send us a note!

<form name="form1" method="get" action="thank_you.asp">

  id: <input type="text" name="id" /><br />
  name: <input type="text" name="name" /><br />
  <br />
  <input type="submit" name="submit" value="submit">

</form>

<!-- thank_you.asp -->

Thank you <strong><%=Request.QueryString("name")%></strong>.<br />
Your ID is: <strong><%=Request.QueryString("id")%></strong><p>
```

When the page is submitted, the form fields will be appended to the querystring of the action page as URL parameters.

It's as simple as that. You change the value associated with your `Request.QueryString` depending on the URL parameter you want to display. If the URL parameter has an attribute called `name`, then you can display its value with `Request.QueryString("name")`. URL parameters will be extremely important when you begin to display database records on your pages. You will usually need to pass that data to other pages.

Sending form values to e-mail

E-mail can add new levels of functionality to a dynamic website. We normally think of e-mails in the context of sending our friends and family members messages using web-based services such as Hotmail or Yahoo!, or using an e-mail client such as Outlook Express, Eudora, or Thunderbird. But how would you feel about sending e-mail from your own ASP pages? How about allowing users to send e-mail from your ASP pages?

E-mail gives your dynamic website that extra edge—you can send e-mail to your users, allow users to send e-mail to you, and allow users to send e-mail to each other. You can use e-mail to help you keep track of what is happening on your site. Let's take a closer look. In the sections that follow, we'll show you how to set up your SMTP server to safely send and receive mail, introduce ASP mail components, discuss some general things to keep in mind about sending mail, and demonstrate how to send mail using three different free components.

Setting up your SMTP server

E-mail is handled by a Simple Mail Transfer Protocol (SMTP) server. Two SMTP servers connect to each other: one server sends the e-mail, while the other accepts the e-mail. You need to have your SMTP server up and running locally before it executes mail.

> *Personal Web Server (PWS), found on Windows 98, does not have an SMTP server, but IIS comes with its own SMTP server. You can shut it down or start it on your own.*

If you will be testing your website's pages locally, you need to configure your SMTP server. However, if you will complete your website's pages and simply upload them to your web host's server, then you can just use your host's SMTP server without worrying about configuring the SMTP server locally, which is a good backup plan if you do run into any trouble configuring your SMTP server locally.

To make sure your SMTP server is configured properly, follow these steps:

1. Open the Control Panel and select Administrative Tools ➤ Internet Services Manager.
2. Expand the plus sign next to the computer icon and the machine name of your PC.
3. Right-click Default SMTP Virtual Server and select Properties.

4. On the General tab, set the IP address field to (All Unassigned) (Figure 5-9).

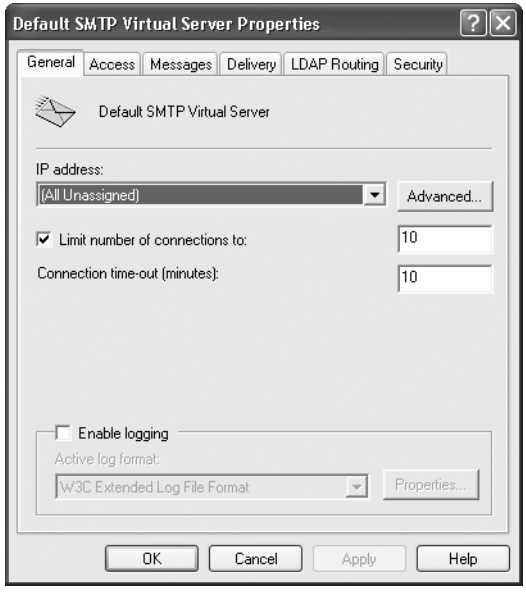


Figure 5-9. Setting the default SMTP virtual server on the General tab

5. Switch to the Access tab and click the Authentication button. Select all three sections: Anonymous access, Basic authentication, and Integrated Windows Authentication (Figure 5-10). Click OK.

Authentication
Select acceptable authentication methods for this resouce.
Anonymous access
No user name or password required.
Basic authentication
The password will be sent over the network in clear text using standard commands.
Requires TLS encryption
Default domain:
Integrated Windows Authentication
The client and server negotiate the Windows Security Support Provider Interface.
OK
Cancel
Help

Figure 5-10. Selecting authentication methods

6. Click the Connection tab. Select the Only the list below radio button and specify the IP address of your local host (Figure 5-11). This will allow your computer to connect only to the SMTP server. Click OK.

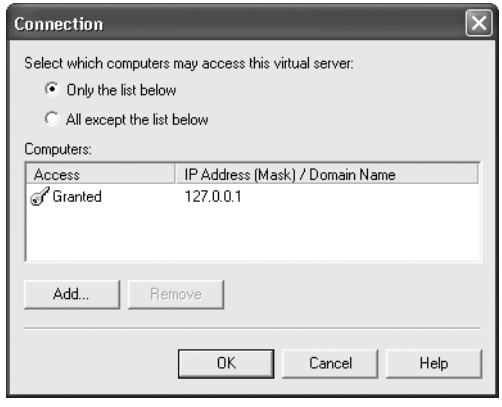


Figure 5-11. Selecting the IP addresses that can connect to the SMTP server

7. Click the Relay Restrictions tab. Select the Only the list below radio button and specify the IP address of your local host. Select the option at the bottom of the dialog box to allow relay for all computers that authenticate (Figure 5-12). Click OK, click Apply, and then close the dialog box.

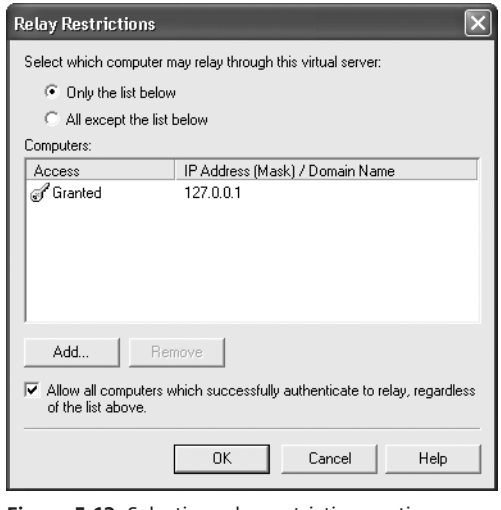


Figure 5-12. Selecting relay restrictions options

The SMTP server can relay e-mail messages by forwarding mail addressed to any e-mail domain. If this feature is accessible to Internet users other than you, then users with malicious intentions can use your SMTP server to forward unsolicited e-mail to thousands of people. By selecting the computers

that may relay through your server, you prevent computers besides your own IP address from relaying messages through your SMTP server.

Your SMTP server should now be configured to send e-mail successfully. Currently, your IP address is the only one allowed to connect to and relay messages from your SMTP server. Feel free to change these options later on by adding more IP addresses or restricting specific ones through the Connection and Relay Restrictions options. When adding or restricting IP addresses, be sure to utilize both options to fully secure your SMTP server.

> *If you encounter any problems with your SMTP server, be sure to check that its settings are correct. Here is a link to a useful web page that lists some of the causes and resolutions of common SMTP server errors:* `http://www.aspplayground.net/forum/m_375481/tm.htm`.

> *For the purposes of this chapter, try to use an e-mail account that you can check easily, as you'll be sending a lot of mail. For example, you may consider using Outlook Express to send and receive e-mail for the exercises in this chapter. You can then leave the Outlook Express program loaded, so you are able to check the e-mail right away.*
>
> *If you have any problems with triggering e-mail in any of these examples locally, however, I suggest you send them remotely using your web host. In the next section, you'll learn how to configure CDO to connect to a remote SMTP server in lieu of your local SMTP server. To use your remote SMTP server, you will need the IP address of your website, the account username and password for one of your e-mail addresses related to your domain name, all of which can be obtained from your web host.*

ASP mail components

ASP components are created by companies or individual web developers to make certain tasks easier, such as uploading or adding other functionality. They are written in a particular language, are packed and installed on a server, and can be accessed to perform particular actions through simple properties. At the moment, we're especially interested in using **mail components** that will allow us to send mail through our SMTP server. Later in the chapter, we'll show you how to use three of the most popular free versions of ASP mail components.

Before you begin using the mail components, there is something very important to note: mail components have to be installed and configured on a server like any other ASP component. If you test your pages on your own local IIS, you can download, install, and begin using all of the mail components. Many hosting companies offer access to a particular mail component to their clients, while some provide none, and still others provide a selection from which to choose. Before signing up with a hosting company, you should make sure that you'll have access to the mail component of your choice. If you don't, you won't be able to trigger mail from your ASP pages.

The good news is that all Windows servers come with a free mail component, formerly known as **Collaboration Data Objects for Windows NT Server** (CDONTS). CDONTS has been phased out since Windows 2000, so if your host uses Windows Server 2003, you must use **CDOSYS**, also known as CDO, which now comes installed with all Windows servers. CDOSYS allows web developers to send e-mail without having to install a third-party mail component and to create much more sophisticated e-mails than ever before, which is why previous versions have simply been phased out.

General requirements for mail components

In order to send an e-mail, of course, certain values are required: sender, receiver, subject, and body. When using a mail component, you usually need to specify the IP address of the SMTP server. Beyond these values, there are other syntactical requirements such as creating, executing, and closing the object. You create the object, provide the values required for the content of the e-mail, execute the object, and finally close the object. Each component has its own syntax, but for the most part they are all very similar with respect to what has to be done. Additionally, each component has many properties, but you will only use those that are required for sending the e-mail. When you decide you want to explore the other properties that the particular mail component you're using has to offer, you should check the manual at the company's website.

You should be aware of the fact that if you will be testing the mail component examples on a server on which you don't have access to administrative options, you may find that your web host will not allow you to send e-mails that are not hosted by that particular SMTP server. In other words, if within the syntax of your mail component you try to send e-mail from `omar@dmxfire.com`, while the server you are sending it from does not host `www.dmxfire.com`, it will reject the e-mail not send anything. This might drive you crazy, especially when you start learning about dynamic e-mail, because you are likely to want to allow your users to send e-mails using their own e-mail addresses (whose domains will probably not be hosted by the server you'll be using), but those will be rejected if your host disallows anonymous domains. You will be stuck with using your own e-mail address with a domain that is local to the server. This is inconvenient because you won't know the e-mail address of the sender since you will have had to resend every e-mail from your own e-mail address. There are some not-so-slick ways around this, which we'll get into later, but for now just know you need to take this issue into account.

Although it can be annoying, there are valid reasons why a hosting company would disallow anonymous domains. Some malicious clients can use the server to send junk mail using an anonymous e-mail address from their web host's very own server. Web hosts disallow anonymous domains access to their SMTP server to protect themselves and others from these crude e-mail tactics. Although you allowed anonymous domains in the preceding section, this is why you specified that only your own IP address have access to the SMTP server.

> *If you will be testing the examples from your local IIS, you are good to go and can use any e-mail address as the sender so long that you adjust some administrative options for your SMTP server, as shown in the "Setting up your SMTP server" section. However, if you will be testing the examples from a server hosted by a web hosting company that disallows anonymous domains to send e-mail from their SMTP server, you need to either contact the host or use a local e-mail address before you start troubleshooting why the e-mail is not being sent. Unfortunately, there is no other way around this.*

Sending e-mail with mail components

In this section, we'll demonstrate how to send mail through three free ASP mail components: CDO, AspEmail, and JMail. For each component discussed, before we explore dynamic e-mail interaction, we'll start off by sending simple, static e-mail (i.e., e-mail with preset values).

Sending e-mail with CDO

CDO is one of the most popular ASP mail components because it comes with IIS (i.e., it is installed by default). Again, components are objects on the server and will be treated as objects in our ASP pages. Every object has properties that you need to define values for.

Let's take a look at CDO syntax. Create a new page and save it as `execute_cdo.asp`. In Code view, type the following above the `<html>` tag, specifying your own e-mail addresses for the `From` and `To` properties. You can also create your own `Subject` and `Body` content.

```
<%
Set myMail = CreateObject("CDO.Message")
myMail.Subject = "Sending email with CDO"
myMail.From = "<EMAIL ADDRESS>"
myMail.To = "<EMAIL ADDRESS>"
myMail.TextBody = "This is a message."
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 1
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserver") = "localhost"
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpauthenticate") = 1
myMail.Configuration.Fields.item("http://schemas.microsoft.com/cdo
/configuration/smtpserverpickupdirectory") =
➥ "c:\inetpub\mailroot\pickup"
myMail.Configuration.Fields.Update
myMail.Send
set myMail = nothing
%>
```

As you can see, the code is very simple. The following creates an instance of the CDO object within your ASP page:

```
Set myMail = CreateObject("CDO.Message")
```

We named the CDO object `myMail`, but you could call it whatever you like: `objSendMail`, `Mail`, `CDOmail`, and so on. Each object property is referred to in the context of its object (i.e., `objCDO.PropertyName`). If you named the object `objSendMail`, for example, you would refer to a property as `objSendMail.PropertyName`.

The preceding code contains four properties and one method: the `From`, `To`, `Subject`, and `TextBody` properties, and the `Send` method. As you might guess, the `From` property contains the e-mail address of the sender, and `To` contains the e-mail address of the receiver. `Subject` contains the subject of the e-mail, and `TextBody` contains the body of the e-mail. Lastly, the `Send` method actually sends the e-mail.

The `Configuration` field is a namespace that defines most fields for the CDO object. These namespaces are stored on the Microsoft website, and you retrieve them in the code to set various configurations for the object, such as whether the object should be sent from a local or remote server, the IP address, username and password authentications, the port, and so on:

```
Configuration.Fields.Item("http://schemas.microsoft.com/cdo/
➥ configuration/sendusing")
```

After the object does its job, you clear it from the server by destroying the object with the following code:

```
set myMail = nothing
```

You can test the `execute_cdo.asp` page by loading it into your browser. Once the page loads into the web browser successfully, the CDO code will have just executed, since the ASP code is executed on the server first and then the page is sent to the user's web browser. The HTML page itself will come up blank, as you haven't done anything else to it. To ensure that the e-mail was sent, you must check the inbox of whatever e-mail address you defined in the `To` property. You should see the following as the body of your e-mail:

Hi, I was sent because the page execute_cdo.asp on my server was accessed.

To send e-mail using CDO remotely, you can change the values of the namespace to send the e-mail through a remote SMTP server:

```
<%
Set myMail = CreateObject("CDO.Message")
myMail.Subject = "Sending email with CDO"
myMail.From = "<EMAIL ADDRESS>"
myMail.To = "<EMAIL ADDRESS>"
myMail.TextBody = "This is a message."
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 2
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserver") = "<IP ADDRESS>"
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpauthenticate") = 1
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusername") = "<USERNAME>"
```

```
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendpassword") = "<PASSWORD>"
myMail.Configuration.Fields.Update
myMail.Send
set myMail = nothing
%>
```

Notice the additional namespaces where you configure the username and password to connect to the SMTP server. Usually, one of the live e-mail accounts associated with your website and its related password would serve as the username and password. Your web host can provide you with this information, as well as the correct IP address.

Formatting e-mail with HTML in CDO In the preceding code, you used the property for plain text so all text defined for the `TextBody` property was translated as plain text and any HTML contained within it will not be parsed.

To use HTML tags in the body of the e-mail, change the `TextBody` property to `HTMLBody`, for example:

```
myMail.HTMLBody = "<html><body>This is a <b>message</b>.</body></html>"
```

Once you have changed the e-mail format to HTML, you can format the content with any of the HTML tags. For example, you could change the font and even add images. Let's say your logo (or any other image) exists in a folder on your server (with the URL path `http://www.dmxfire.com/images/logo_main.gif`), and you want to embed it in an e-mail with the `<img>` tag. You can now embed this link as the value of the `HTMLBody` property, and the HTML will be parsed if the recipient allows HTML e-mails:

```
myMail.HTMLBody = "<img
➥ src='http://www.dmxfire.com/images/logo_main.gif'>"
```

> *As you recall from Chapter 3, double quotes mark the start and end of a variable value, so another set of quotations within those will confuse the ASP processor and cause erroneous results. You need to escape the double quotes in the HTML tags. You can escape quotes that may need to appear with the value of a variable by using either two double quotes, "", or one single quote, '.*

When you check the e-mail, you should see the image in the body. Now you can understand how many big websites send out those newsletters that look like web pages to your inbox! You can add any HTML you like to produce a web page as the body of an e-mail—you could even attach a CSS stylesheet that's hosted remotely.

But before you decide to send all your e-mails in HTML, consider the fact that some of your recipients' mail programs may disallow HTML to render in e-mails due to the security risks involved. HTML tags will not be parsed in an e-mail program that rejects HTML; in such a case, the e-mail will just appear with unrendered HTML markup, making the e-mail completely unreadable. Plain text e-mails work fine and are the best format for the majority of your e-mails, unless you feel it's absolutely necessary to send an e-mail in HTML.

Sending carbon copies (cc) and blind carbon copies (bcc) with CDO You may be wondering how to send an e-mail to several e-mail addresses. To do this, you can simply repeat the `objCDO.To` property, inputting a different e-mail address. For example, here's how you could change the code for the `execute_CDO.asp` page to send your e-mail to two different addresses:

```
myMail.To = "<EMAIL ADDRESS>"
myMail.Cc = "<EMAIL ADDRESS>"
```

Alternatively, to send a carbon copy of the e-mail to another address, use the cc and bcc properties, `myMail.Cc` and `myMail.Bcc`:

```
myMail.To = "<EMAIL ADDRESS>"
myMail.Cc = "<EMAIL ADDRESS>"
myMail.Bcc = "<EMAIL ADDRESS>"
```

In the first example, both recipients will see the e-mail addresses of all that receive carbon copies, but in the second example, no recipient will be able to see the e-mail address of the recipient who receives the blind carbon copy. However, the bcc recipient will still see the e-mail addresses of the cc recipients.

> *The IMessage Interface documentation at Microsoft provides more information on CDO's interface:* `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cdosys/html/39186eaa-c4c1-430a-9715-35e291925c5c.asp`. *We also recommend perusing the various levels of that documentation for a deeper look at CDO for Windows 2000.*

In the next couple of sections, we'll cover the syntax of two other popular mail components: AspEmail and JMail. However, we won't go into as much detail for AspEmail and JMail as we have for CDO in this section. It isn't that one is better than the others—it's just that you can do the same things with each component once you know the basic syntax (especially the properties).

Sending mail with AspEmail

AspEmail is another popular ASP mail component, but unlike CDO it doesn't come with IIS. It is proprietary software available from Persits Software, Inc. (`www.persits.com`), which offers a free version of this mail component on its website as well as some commercial premium features. When you download the free version of AspEmail, the commercial premium features will be available for 30 days, but the generic features will always be available. The commercial version comes with a few more useful properties that you can use, but the free version contains all that you need to work through the examples in this chapter, and possibly for most of your own examples down the road.

You can download the latest version of the AspEmail component at `www.aspemail.com/download.html`. Once you download the component, install it by double-clicking the file. Make sure you register the component when prompted during installation. If you will be using this component on your website that is hosted remotely, make sure your web host has this component installed on the server available for your use. Usually when you sign up for a web hosting package, you are informed about which components come with the package.

Back in Dreamweaver MX, create a new dynamic ASP web page and save it as execute_aspemail.asp. Add the following code above the <html> tag:

```
<%
Set objASPEmail = Server.CreateObject("Persits.MailSender")

objASPEmail.Host = "<IP ADDRESS>"
objASPEmail.From = "<EMAIL ADDRESS>"
objASPEmail.AddAddress "<EMAIL ADDRESS>"

objASPEmail.Subject = "Test ASPEMAIL"
objASPEmail.Body = "Hi, this is an AspEmail component test."

objASPEmail.Send
Set objASPEmail = Nothing
%>
```

Replace the text <EMAIL ADDRESS> set in the From and AddAddress properties with the appropriate e-mail address. The From property holds the sender's e-mail address, and the AddAddress property holds the recipient's e-mail address.

Just like with the CDO object, you have to create an instance of the object:

```
Set objASPEmail = Server.CreateObject("Persits.MailSender")
```

For the AspEmail component, you need to specify the SMTP server address in a property called Host, something you didn't have to do with CDO when running it locally. If you are testing locally, you should input your IP address. (You can find out your IP address by typing ipconfig at the command prompt or by simply using 127.0.0.1.) If you are executing the page on a page hosted by a web host, you need to contact the host for the correct SMTP server address, as mentioned previously.

The other properties are not new to you. The only differences are the property names, but they all have same functionality as the CDO properties. However, you may be confused by the AddAddress method earlier. This is where you specify the recipient's e-mail address, and it is no different than the To property of CDO. You can also specify a label for the AddAddress property, displaying a name for the e-mail address. To do so, add a comma after the e-mail address and put the name within double quotes:

```
objASPEmail.AddAddress "<EMAIL ADDRESS>", "omar"
```

Also, notice that to define a value for the AddAddress method in AspEmail, you don't have to use an equal sign; that is because it is a method, not a property, here. It's a good idea to keep an example of each mail component's syntax on your PC for quick reference.

> *The* AddAddress *does not use an equal sign because the e-mail address is being passed to a method. Properties will always be set using the equal sign, whereas methods will not.*

If you execute the page on a live server and check the e-mail for the e-mail address you specified, you should see the following in the body of the e-mail:

Hi, this is an AspEmail component text.

Formatting e-mail with HTML in AspEmail By default, AspEmail sends mail in plain text format. You can send AspEmail e-mail in HTML format by defining the following property as true: `IsHTML = True`. (If this property is defined as `False`, AspEmail sends e-mail in plain text.)

Change the code for the `execute_aspemail.asp` page to the following to send an e-mail in HTML format:

```
<%
Set objASPEmail = Server.CreateObject("Persits.MailSender")

objASPEmail.Host = "<IP ADDRESS>"
objASPEmail.From = "<EMAIL ADDRESS>"
objASPEmail.AddAddress "<EMAIL ADDRESS>"

objASPEmail.Subject = "Test ASPEMAIL"
objASPEmail.Body = "Hi, this is an <b>AspEmail</b> component test."
objASPEmail.IsHTML = True

objASPEmail.Send
Set objASPEmail = Nothing
%>
```

If you execute the page on a live server and check the e-mail for the recipient e-mail address you specified, you should see the following in the body of your e-mail:

Hi, this is an **AspEmail** component text.

Sending carbon copies (cc) and blind carbon copies (bcc) with AspEmail AspEmail also has a method for sending carbon copies, called `AddCc`. To use this method, make the following changes to the `execute_aspemail.asp` page:

```
<%
Set objASPEmail = Server.CreateObject("Persits.MailSender")

objASPEmail.Host = "<IP ADDRESS>"
objASPEmail.From = "<EMAIL ADDRESS>"
objASPEmail.AddAddress "<EMAIL ADDRESS>"
objASPEmail.AddAddress "<EMAIL ADDRESS>"
objASPEmail.AddCc "<EMAIL ADDRESS>"
objASPEmail.Subject = "Test ASPEMAIL"
objASPEmail.Body = "Hi, this is an <b>AspEmail</b> component test."
objASPEmail.IsHTML = True

objASPEmail.Send
Set objASPEmail = Nothing
%>
```

As with the AddAddress method, you don't use an equal sign when giving a value to the AddCc method. Execute the page on a live server and check both e-mail addresses. A copy of the e-mail should have been sent to both.

You can also do the same thing by repeating the AddAddress method:

```
objASPEmail.AddAddress "<EMAIL ADDRESS>"
objASPEmail.AddAddress "<EMAIL ADDRESS>"
```

For blind carbon copies, you use the AddBcc method:

```
objASPEmail.AddBcc "email address"
```

AspEmail extras AspEmail has a couple of extra properties and methods you can use: You can define a display name for the sender e-mail address by using the FromName property, and by using a method called AddReplyTo you can specify a different e-mail address to be used when the recipient clicks the Reply button for that e-mail. Here are some examples:

```
objASPEmail.From = "email address"
objASPEmail.FromName = "omar"
objASPEmail.AddReplyTo "different email address"
```

For more information on AspMail's various properties, see the component's online manual at www.aspemail.com/manual.html.

Sending e-mail with JMail

Another popular proprietary mail component is JMail from Dimac Development (www.dimac.net). Look for the product named w3JMail. You can download the free w3JMail version 4.5 from the Dimac website, along with many other commercial products, and install it by simply double-clicking the file after a successful download. (Note that you may be prompted to fill out a registration form.) If you will be using this component on your website that is hosted remotely, make sure your web host has this component installed on the server and available for your use.

Back in Dreamweaver, create a new dynamic ASP web page and save it as execute_jmail.asp. Add the following code above the <html> tag:

```
<%
Set JMail  = Server.CreateObject("JMail.SMTPMail")

JMail.ServerAddress = "<IP ADDRESS>"
JMail.Sender = "<EMAIL ADDRESS>"
JMail.AddRecipient "<EMAIL ADDRESS>"

JMail.Subject = "Test JMAIL"
JMail.Body = "Hi, this is a JMail component test."

JMail.Execute
Set JMail = Nothing
%>
```

Replace the text <EMAIL ADDRESS> set for the Sender and AddRecipient properties with the appropriate e-mail addresses. The Sender property holds the sender's e-mail address and the AddRecipient property holds the recipient's e-mail address. In the ServerAddress property, you will need to specify your local IP address or, if testing remotely, the IP address of your website, which you can obtain from your web host.

Execute the page on your live server. When you check the e-mail, you should see the following in the body:

Hi, this is a JMail component test.

Formatting e-mail with HTML in JMail You can change JMail e-mail from plain text format to HTML by defining the ContentType property as "text/html". Change the code for the execute_aspemail.asp page to the following to send an e-mail in HTML format:

```
<%
Set JMail  = Server.CreateObject("JMail.SMTPMail")

JMail.ServerAddress = "<IP ADDRESS>"
JMail.Sender = "<EMAIL ADDRESS>"
JMail.AddRecipient "<EMAIL ADDRESS>"

JMail.Subject = "Test JMAIL"
JMail.Body = "Hi, this is a <b>JMail</b> component test."
JMail.ContentType = "text/html"

JMail.Execute
Set JMail = Nothing
%>
```

Execute the page on a live server and check the e-mail for the recipient e-mail address you specified. You should see the following in the body of your e-mail:

Hi, this is a **JMail** component text.

Sending carbon copies (cc) and blind carbon copies (bcc) with JMail You can also send mail to several different e-mail accounts using the AddRecipientCc method or by simply repeating the AddRecipient property. Here are two examples:

```
JMail.AddRecipient "<EMAIL ADDRESS>"
JMail.AddRecipientcc "<EMAIL ADDRESS>"
```

and

```
JMail.AddRecipient "<EMAIL ADDRESS>"
JMail.AddRecipient "<EMAIL ADDRESS>
```

Sending a bcc to another e-mail address is also easy—you just need to add the `AddRecipientBcc` property. Here is an example:

```
JMail.AddRecipient "<EMAIL ADDRESS>"
JMail.AddRecipientBcc "<EMAIL ADDRESS>"
```

For more specific information about JMail and its various properties, please visit the Products section of the Dimac site. You can also find the JMail manual on the Products page for w3JMail v4.5.

Dynamic e-mail interaction

It's time for some action! Since everyone is moving from static to dynamic web pages these days, let's take that attitude with our e-mail also. You now know how to send e-mail through ASP pages, but only how to send it once a page loads and with predefined values. How about sending it after a specific action occurs or allowing the user to define the values for the properties? This is what we will refer to as "dynamic" e-mail, as property values are not hard-coded; rather, they are ever changing.

Once you set up the web application, users of your website will be able to alter the property values. How much control you want users to have is up to you. You can allow users to specify the sender e-mail address and predefine the recipient. This is usually the case with feedback or contact forms. You can also allow users to define the sender, subject, and body. The properties will act as regular variables.

> *In the "Sending e-mail with mail components" section, you learned how to use three different mail components: CDO, AspEmail, and JMail. For the rest of this chapter's examples, you will use the CDO mail component, because it is more accessible since it comes installed with IIS. Now that you have been exposed to the syntax of all three components, though, you should feel free to modify the code for the mail component of your choice.*

In this section, we'll first cover how to send mail by hyperlink and via a form button, and how to make e-mail property values dynamic.

Sending e-mail by hyperlink

You can trigger an e-mail based on many different conditions. Let's start by triggering an e-mail based on a hyperlink click. In Dreamweaver, open a new document and enter the following script, saving it as `email_hyperlink.asp`:

```
<%
'Check if hyperlink was clicked
If Request.QueryString("link") = "true" Then

Set myMail = CreateObject("CDO.Message")
myMail.Subject = "Sending email with CDO"
myMail.From = "<EMAIL ADDRESS>"
```

```
myMail.To = "<EMAIL ADDRESS>"
myMail.TextBody = "This is a message."
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 1
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserver") = "localhost"
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpauthenticate") = 1
myMail.Configuration.Fields.item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverpickupdirectory") =
➥ "c:\inetpub\mailroot\pickup"
myMail.Configuration.Fields.Update
myMail.Send
set myMail = nothing

'Redirect to same page adding message in querystring
Response.Redirect("email_hyperlink.asp?email=sent")

End If
%>

Type the following code within the <body> tag:

<a href="email_hyperlink.asp?link=true">send email</a>
```

Once the page is loaded, nothing happens except that the hyperlink is displayed—but notice the new querystring attached. That lets you know that the mail code was executed, because in the preceding code you attached the querystring based on conditional logic that the code was executed (Figure 5-13).

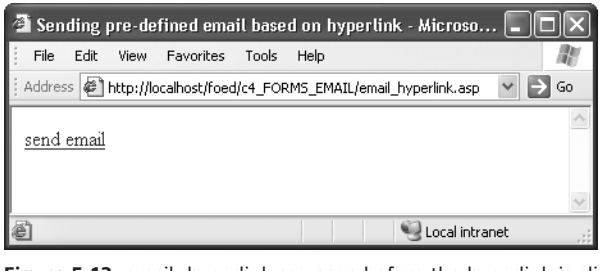


Figure 5-13. email_hyperlink.asp page before the hyperlink is clicked

The e-mail is not triggered automatically because it is embedded in a conditional statement that tells it to execute on the condition that a URL parameter named `hyperlink` exists and equals `true`. When the page is first loaded, there won't be any URL parameters. Clicking the hyperlink takes you back to the same page, but this time passing the appropriate URL parameter, which triggers CDO to send

the e-mail. After CDO is executed, it redirects you back to the same page with a new URL parameter, letting you know that the CDO was executed (Figure 5-14).

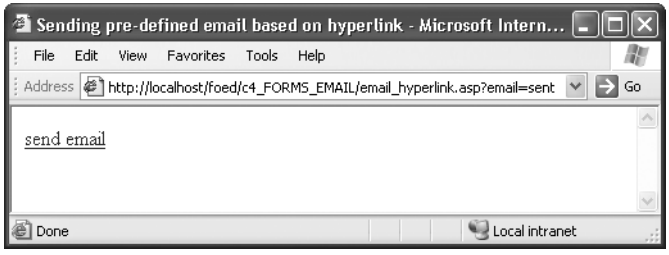


Figure 5-14. email_hyperlink.asp page after the hyperlink is clicked

One of the cool things about this method is that everything is on a single page, yet it behaves as though two pages were used. The ASP script is not executed even if the page is consecutively refreshed; it executes only if that particular hyperlink is clicked or the correct URL parameter is passed. The e-mail is executed only when a querystring named `hyperlink` equals `true`. After the querystring is passed from the hyperlink, the e-mail is executed, the user is redirected to the same page using `Response.Redirect`, and the querystring is cleared. For this reason, the e-mail executes only if the hyperlink is clicked.

Another way to do this is to have the ASP script on a separate page. You could link to that page and redirect back or to another page, but in this case, the e-mail will be sent whenever the page with the CDO is accessed, whether intentionally or not. At any rate, you could always wrap conditional statements around your ASP script, making sure it is executed only if certain criteria are fulfilled.

Sending e-mail via a form button

Sending e-mail based on the click of a form button is fairly simple. To demonstrate, create a new dynamic ASP web page and save it as `email_frmbutton.asp`, and then follow these steps:

1. Select Insert ➤ Form ➤ Form.
2. Place your cursor inside the form and select Insert ➤ Form ➤ Button.
3. For Accessibility Attributes, select no label and click OK.
4. Highlight the button in the Dreamweaver document window, and from the Properties panel change the name of the button to `email_submit` and change the button value to `Send Mail`. Have the action of the form submit back to this same page. The form's method should be `post`, not `get`. (`get` will send the form values as URL parameters.)
5. Add the following code above the `<html>` tag:

```
<%

If Request.Form("email_submit") <> "" Then

Set myMail = CreateObject("CDO.Message")
myMail.Subject = "Sending email with CDO"
myMail.From = "<EMAIL ADDRESS>"
```

```
myMail.To = "<EMAIL ADDRESS>"
myMail.TextBody = "This is a message."
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 1
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserver") = "localhost"
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpauthenticate") = 1
myMail.Configuration.Fields.item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverpickupdirectory") =
➥ "c:\inetpub\mailroot\pickup"
myMail.Configuration.Fields.Update
myMail.Send
set myMail = nothing

Response.Redirect "email_frmbutton.asp?email=sent"

End If

%>
```

The conditional statement surrounding the ASP script checks to see if the form field named `email_submit` is empty. Once the form is submitted, this form field (the Submit button) will have a value so the ASP script executes. (Please see Chapter 3 for a review of VBScript syntax.)

6. Test the page live and click the form button to send the e-mail (Figures 5-15 and 5-16).

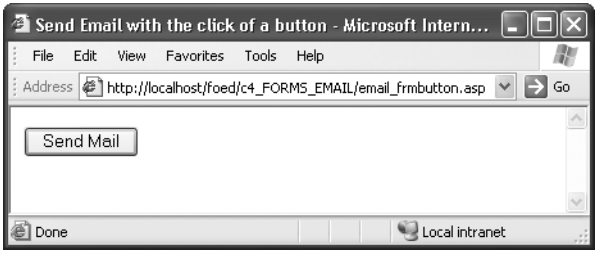


Figure 5-15. email_frmbutton.asp in a web browser before the Send Mail button is clicked

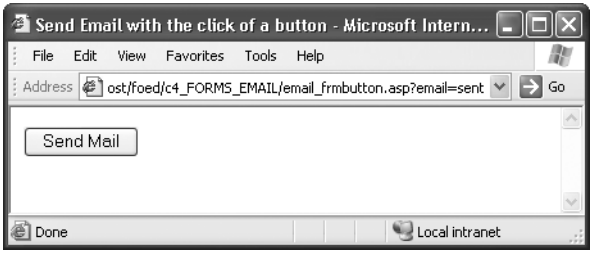


Figure 5-16. email_frmbutton.asp in a web browser after the Send Mail button is clicked. Notice the querystring.

Sending e-mail based on your own conditions is the first step in sending dynamic e-mail. You can also send e-mail based on other conditions, such as an event that might occur within the ASP page. For example, you may want to send yourself an e-mail whenever someone accesses a particular page on your website. Once you understand the concept, it is up to you to determine how to best apply it.

Making e-mail property values dynamic

So far, you've been sending e-mails with predefined values. The next step is to make the e-mail property values themselves dynamic, and not just the execution. An example where this is useful is when you want users to send e-mail from your website to you or to other users. You might also simply want to collect dynamic values from users. For example, web users could send you feedback from a page on your site, and you could collect their IP address as they send you feedback. Web users can send other web users a link from your website. You will look at these applications in the following sections.

Sending e-mail with dynamic values

Let's now add some dynamic values to your e-mails. In this section, you'll collect the IP address of the sender of an e-mail and the page the user sent the e-mail from in the body of your e-mail. To do so, first create a new dynamic ASP web page and save it as `dynamic_values_in_email.asp`. Create the same form as in the previous example and follow these steps:

1. Select Insert ➤ Form ➤ Form.
2. Place your cursor inside the form and select Insert ➤ Form ➤ Button.
3. For Accessibility Attributes, select no label and click OK.
4. Highlight the button in the Dreamweaver document window, and from the Properties panel change the name of the button to `email_submit` and change the button's value to `Send Mail`. Have the action of the form submit back to this same page. The form's method should be `post`, not `get`.
5. Add the following code above the `<html>` tag:

```
<%

If Request.Form("email_submit") <> "" Then

Set myMail = CreateObject("CDO.Message")
myMail.Subject = "Form button clicked"
myMail.From = "<EMAIL ADDRESS>"
myMail.To = "<EMAIL ADDRESS>"
myMail.TextBody = "Here is the IP Address of the sender: " &
➥ Request.ServerVariables("REMOTE_HOST") & vbcrlf &
➥ "This is the page
➥ the sender sent the page from: " & Request.ServerVariables("URL")

myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 1
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
```

```
➥ /configuration/smtpserver") = "localhost"
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpauthenticate") = 1
myMail.Configuration.Fields.item("http://schemas.microsoft.com/cdo/
➥ configuration/smtpserverpickupdirectory") =
➥ "c:\inetpub\mailroot\pickup"
myMail.Configuration.Fields.Update
myMail.Send

set myMail = nothing

Response.Redirect "dynamic_values_in_email.asp?email=sent"

End If

%>
```

6. Test the page live and click the form button to send the e-mail.

The code is basically the same as in the previous example, except that you've added some dynamic variables into the body of the e-mail. You also changed the value of the redirection page to the new page. To create the body of the e-mail with strings and dynamic variables, you have to concatenate them together. (If you need a refresher on concatenation, see Chapter 3.)

> *If you leave out a concatenation character, you may break the value, so take care to concatenate the strings and variables together properly.*

You collected two environment variables: the IP address and the URL to the current page. The only new code in the body is `vbcrlf`. This is simply a built-in VBScript string constant that creates a carriage return and line feed. If you want to use HTML tags such as `<br>` instead, you have to change the e-mail format to HTML format using the properties described earlier in this chapter.

You have now added dynamic variables inside the body of your e-mail. You could add the variables into the Subject line, or you could also retrieve other environment variables. Figure 5-17 shows an example of the e-mail the recipient will see.

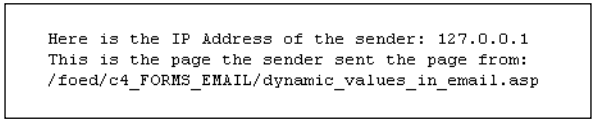

Here is the IP Address of the sender: 127.0.0.1
This is the page the sender sent the page from:
/foed/c4_FORMS_EMAIL/dynamic_values_in_email.asp

Figure 5-17. Text specified in an ASP page as it arrives in an e-mail address's inbox

Sending e-mail with dynamic form field values

Now let's replace entire property values with dynamic variables instead of hard-coding them. Start off by creating a new dynamic ASP web page and saving it as `send_email_form.asp`. Insert into an empty form three text fields and one text area, to parallel the properties of the CDO properties `.To`, `.From`, `.Subject`, and `.TextBody`. Rename the text fields and add labels and properties as shown in the following table.

Text field	Label	Properties
`txtsender`	Sender Email:	
`txtrecipient`	Recipient Email:	
`txtsubject`	Subject:	
`txtbody`	Body:	`TextArea`

Also insert a Submit button, set its value to Send Email, and leave the name of your button as is (Submit). Highlight the form and from the Properties window change the action of the form to be sent back to the same page, making sure the method is set to `post`. The user will insert the CDO property values through this form (Figure 5-18).

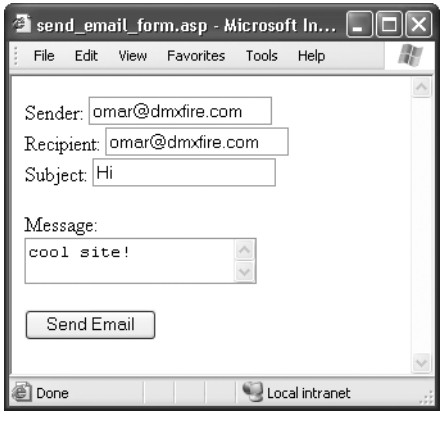


Figure 5-18. send_email_form.asp in a web browser before the form is submitted

> *The convention is to prefix text field names with* txt *so that when you retrieve them later you recognize quickly that the names stand for form fields, but you do not always have to do this. Feel free to use whatever convention works for you and makes it easiest for you and others to read the code.*

Switch to Code view and type the following ASP script above the <html> tag. You will be using Request.Form to retrieve the values of the form fields.

```
<%
If Trim(Request.Form("submit")) <> "" Then

Set myMail = CreateObject("CDO.Message")

myMail.Subject = Trim(Request.Form("txtsubject"))
myMail.From = Trim(Request.Form("txtsender"))
myMail.To = Trim(Request.Form("txtrecipient"))
myMail.TextBody = Trim(Request.Form("txtbody"))

myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 1
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserver") = "localhost"
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
/configuration/smtpauthenticate") = 1
myMail.Configuration.Fields.item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverpickupdirectory") = "c:\inetpub\mailroot
➥ \pickup"
myMail.Configuration.Fields.Update
myMail.Send

set myMail = nothing

Response.Redirect "send_email_form.asp?email=sent"
End If
%>
```

You need to retrieve the values of the form fields. As you know from Chapter 3, this is done with Request.Form("textfield_name"). Also recall that in order to trim variables from extra spaces left to the right of the value, you can use the VBScript function Trim, like this:

```
Trim(Request.Form("<TEXTFIELD NAME>"))
```

For the From property, you want to retrieve the text field named txtsender, so type txtsender between the double quotes. Follow the same procedure for the rest of the properties. Change the name of the form field to reflect the particular one you want to retrieve for the property value you are defining.

Test the page live, enter appropriate values for each form field, and submit the form. The CDO will have executed once your page reloads. Finally, check the e-mail you entered for the recipient e-mail account.

Sending a page to a friend

From the previous two examples, you've learned enough to do a lot of different tricks. It's just a matter of applying the various concepts to create the web application. Let's now take a look at a complete "send page to friend" application, in which you'll apply the techniques outlined previously. You will hard-code some of the property values and allow users to define other property values. You will also use HTML in the body this time.

You'll create a new dynamic ASP web page with a form very similar to the form created in the previous example. The difference here is that you will predefine the subject and body in this example, since that is usually the case with "send page to friend" applications. It isn't practical to run this example locally, so you will use the CDO code (or the code for the other mail components discussed earlier) for connecting to a remote SMTP server. Upload this page to your remote website so that you can see it used practically.

Create a new dynamic page in Dreamweaver, save it as `send_page_form`, and follow these steps:

1. Create a form with two text fields and a Submit button.
2. Rename the first text field `"txtsender"`, and rename the second text field `"txtrecipient"`.
3. Change the value of the Submit button to `"Send"` and leave the name of the button as is (Submit).
4. Format your form however you like. Make sure the `Action` attribute specifies the correct page that will process the form; in this case, it is the same page that contains the form.
5. Load this page in Dreamweaver MX and switch to Code view. Enter the following ASP script above the `<html>` tag:

```
<%
If Request.Form("Submit") <> "" Then

Set myMail = CreateObject("CDO.Message")
myMail.Subject = "A friend has sent you a dmxfire.com page"
myMail.From = Trim(Request.Form("txtsender"))
myMail.To = Trim(Request.Form("txtrecipient"))
myMail.HTMLBody = "Hi " & Trim(Request.Form("txtrecipient"))
➥ & "!" & "<br>" & Trim(Request.Form("txtsender")) & " thought you
➥ might like to check out this page:<p>" &  "<a href=" & """http://
➥ www.dmxfire.com" & Request.ServerVariables("URL") & """>" &
➥ http://www.dmxfire.com" & Request.ServerVariables("URL") &
➥ "</a>"
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 2
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpauthenticate") = 1
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserver") = "<IP ADDRESS>"
```

```
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusername")= "<USERNAME>"
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendpassword")= "<PASSWORD>"
myMail.Configuration.Fields.Update
myMail.Send
set myMail=nothing

Response.Redirect "send_page_form.asp?email=sent"

End If

%>
```

You simply assign the value of the `txtsender` text field to the `From` property, and the value of the `txtrecipient` text field to the `To` property. Then you predefine the values for the `Subject` and `Body`. Don't let this code confuse you—you concatenated some predefined strings with form fields to produce the statement in the e-mail:

```
Hi <recipient_name>!
<sender_name> though you might like to check out this page:

<url>
```

The only confusing code may relate to how you put together the URL link. You need to produce the markup for the anchor element:

```
<a href="url">url</a>
```

The URL appears twice in the element and is put together with a hard-coded string and an environment variable called `URL`. Because this environment variable does not retrieve the domain name, you have to prepend it with the domain name, like this:

```
"http://<DOMAIN NAME>" & Request.ServerVariables("URL")
```

This code will give you the complete URL, which you can then insert into both places within the HTML anchor element.

Now let's consider the double quotes. Because each double quotes within the value of a variable has to be escaped with another double quotes, you end up with many double quotes. The exact locations where you want to break up the HTML anchor element are up to you; we broke it up in the following blocks:

```
<a href=

"http://localhost

Request.ServerVariables("URL")

">
```

```
http://localhost

Request.ServerVariables("URL")

</a>
```

Not only does each of the preceding blocks need to be concatenated and surrounded in double quotes, but also any double quote that already exists as part of the HTML has to be changed into two double quotes. The following code is produced:

```
"<a href=" & """http://localhost" & Request.ServerVariables("URL")
& """>" & "http://localhost" & Request.ServerVariables("URL") & "</a>"
```

After the CDO executes, you redirect to the same page, adding a URL parameter to inform you that the action has occurred.

Test the page live, add some vales in the form fields, submit the form, and check the e-mail.

Let's make this web application even cooler. You can display a message after the e-mail has been sent informing the sender. You already have a way of letting the user know that the CDO executed, which is the fact that you pass a URL parameter along with the redirection underneath the ASP script. Add the following code to your Dreamweaver document:

```
<% If Request.QueryString("email") = "sent" Then %>
Thank you, the page has been sent.
<% End If %>
```

> *Once the page is refreshed, all values entered in the form elements are lost unless you pass them into a querystring.*

If you really want to display a value that was entered in the form, you can pass those values in the URL querystring when you redirect after sending the e-mail. For example, in the preceding mail code, right after you send the e-mail you can change the `Response.Redirect` value to pass the `txtrecipient` text field value in the querystring. In the mail code, change the following:

```
Response.Redirect "send_page_form.asp?email=sent"
```

to this:

```
Response.Redirect("send_page_form.asp?email=sent&recipientemail=" &
➥ Request.Form("txtrecipient"))
```

You can then retrieve the e-mail from the querystring to display it. Let's change the preceding `If` statement that displays the message to also display the recipient e-mail (see Figure 5-19):

```
<% If Request.QueryString("email") = "sent" Then %>
Thank you, the page has been sent to <%=Request.QueryString
➥ ("recipientemail") %>.
<% End If  %>
```

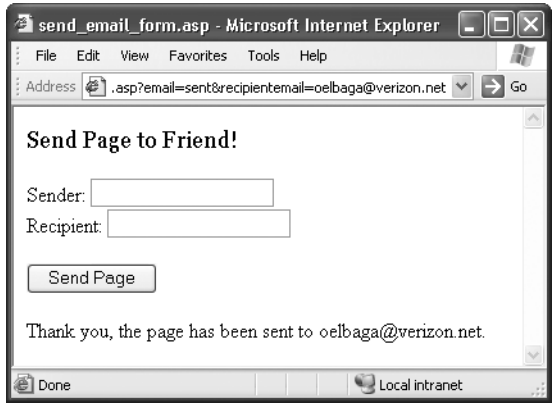


Figure 5-19. send_page_form.asp after the form is submitted

Real-world examples using forms

You've gained a lot of knowledge in this chapter about sending e-mail from ASP web pages, and now you will use that knowledge to create a couple of real-world examples. Although these examples are closely related to forms, they both require a connection to a database to retrieve data.

> *If you are not completely comfortable with databases, we recommend revisiting Chapter 4 before working through these examples.*

Sending a user a forgotten password

With all this e-mailing going on, you may be curious to find out how to e-mail users their passwords when they forget. In this exercise, you'll provide users the option to have their password e-mailed to them when they submit their username to a page on your site. The concept is very simple, and you have all the knowledge necessary to implement it.

You do need to start off with a database that has a table with columns for a username, password, and e-mail address. Make sure your database has some records to use for testing the application.

Now, all you need to do is create a form through which a user can submit his or her username. The username will be submitted to the same page that will filter a recordset, retrieving the record that corresponds to username submitted through a form. You don't need to verify anything at this point, because all we're going to do is send the password information to the e-mail address already stored in the record and associated with that username. You then simply send the e-mail to the e-mail address stored in the record with the password, putting any other recordset values of your choice currently stored in the record, such as contact information.

Create a new dynamic ASP web page, save it as `forgotten_pwd.asp`, and follow these steps:

1. Insert a form with one text field and one Submit button. Call the text field `txtusername` and change the value of the button to `E-mail password`.
2. Change the action of the form to submit back to itself, `forgotten_pwd.asp`, and make sure the method is set to post.
3. Create a connection to a database table that stores the user-related data for your site, including e-mail addresses. From the Application panel, select Databases ➤ Custom Connection String to create a connection to your database.
4. Add a new recordset named `rs_get_pwd` (Figure 5-20). Make sure the entries in the Table and Columns fields match your own database table and columns.

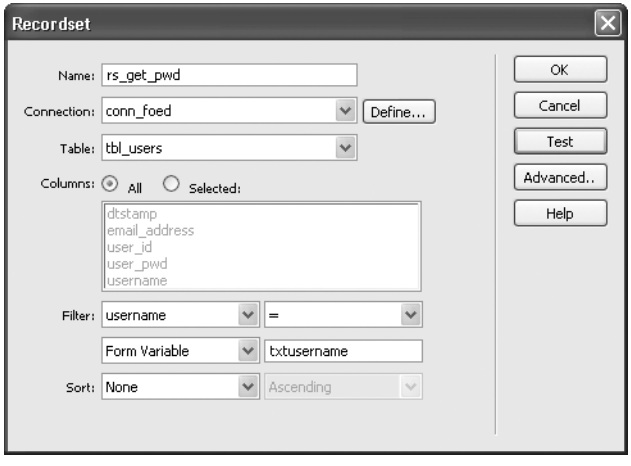


Figure 5-20. The rs_get_pwd recordset after customization

5. Click OK. The recordset will retrieve the `email`, `first_name`, and `user_pwd` columns for the username submitted in the form.
6. Next, you'll add a conditional statement around both the recordset and the ASP script. Add the following ASP script directly *below* the recordset code:

```
<%

If Trim(Request.Form("submit")) <> "" Then

Set myMail = CreateObject("CDO.Message")

myMail.Subject = "Here is your password."

' put an e-mail address associated with your site here
myMail.From = "<EMAIL ADDRESS>"
```

```
'make sure the password column below reflects the name
➥ of the db password column
myMail.To = rs_get_pwd.Fields.Item("<PASSWORD COLUMN>").Value

myMail.TextBody = ""Your password is: " & rs_get_pwd.Fields.Item
➥ ("<PASSWORD COLUMN>").Value

myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 2
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpauthenticate") = 1
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserver") = "<IP ADDRESS>"
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusername")= "<USERNAME>"
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendpassword")= "<PASSWORD>"
myMail.Configuration.Fields.Update
myMail.Send
set myMail=nothing

Response.Redirect "forgotten_pwd.asp?email=sent"
End If

%>
```

Make sure that you change the names within brackets, such as `<PASSWORD COLUMN>`, to reflect your name definitions, both where you need to specify your database columns and server information to trigger e-mails from the SMTP server.

The `To` property is set to the e-mail retrieved from the database record. You then append the database column that stores the password for the record to the body and customize your message. Finally, you redirect and close the `If` statement with `End If`.

We redirect passing a URL parameter until after the e-mail has been sent successfully. It is possible to easily display a related message once the e-mail has been sent by simply checking for the URL parameter's existence. Add the following code anywhere in the body of the page you want the message to appear:

```
<% If Request.QueryString("email") = "sent" Then %>
Congratulations! Your password has been sent to the e-mail address
you registered with. Thanks!.
<% End If %>
```

Your forgotten password web application is complete! Test the page live, insert a real username, and the password will be sent to the e-mail stored in the e-mail column for that particular user (Figure 5-21).

Figure 5-21. forgotten_pwd.asp in the web browser before submission

Creating a mailing list

Before we end this chapter, let's create one more web application that involves sending e-mail to users. In this example, you'll learn how you can send e-mails to your users whenever you need to. The e-mail addresses will be pulled from a recordset, and then you will add the mail component code and have it repeat itself to send the e-mail to each e-mail in the recordset. In the database, you've stored the e-mail address for each user in a column in the `users` table. For this example, you'll pull e-mails from this table and send the same e-mail to each address in one shot.

Create a new dynamic ASP web page, save it as `mailing_list.asp`, and follow these steps:

1. Create a new connection from the Connections menu and select a table that stores email addresses for your users.
2. Create a recordset called `rs_mailinglist_emails`. Highlight only the column that contains the email addresses and click OK (Figure 5-22).

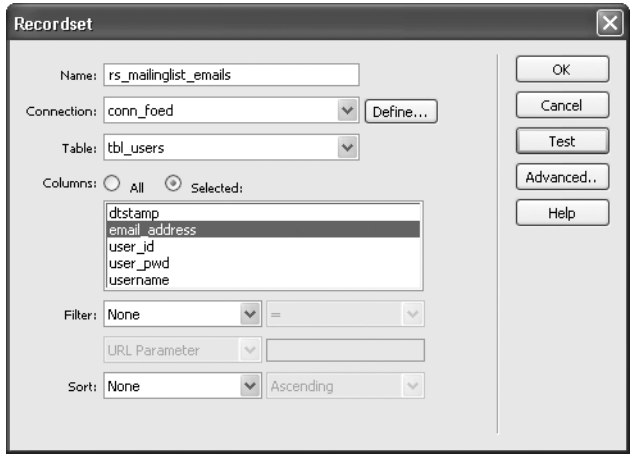


Figure 5-22. The rs_mailinglist_emails recordset in Dreamweaver after its customization

3. Add a form to your page with two text fields and a button. Call the first text field `txtsubject` and the second one `txtbody`. From the Properties window, change the `txtbody` text field to `Multi-line`. Change the value of the submit button to `Send Mail to All Users`.

4. Now switch to Code view and right below the closing `</form>` tag, insert the mail code. Don't forget to make sure that you change the names within brackets, such as `<EMAIL ADDRESS>`, to reflect your name definitions.

```
<%
Set myMail = CreateObject("CDO.Message")

myMail.Subject = Request.Form("txtsubject")

' put an e-mail address associated with your site here
myMail.From = "<EMAIL ADDRESS>"

'make sure password column below reflects recordset name
➥ and name of email column
myMail.To = rs_mailinglist_emails.Fields.Item("<EMAIL COLUMN>").Value

myMail.TextBody = Request.Form("txtbody")

myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusing") = 2
'Server port
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserverport") = 25
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpauthenticate") = 1
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/smtpserver") = "<IP ADDRESS>"
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendusername")= "<USERNAME>"
myMail.Configuration.Fields.Item("http://schemas.microsoft.com/cdo
➥ /configuration/sendpassword")= "<PASSWORD>"
myMail.Configuration.Fields.Update
myMail.Send
set myMail=nothing
%>
```

5. You need to manually change the `To` property to take the e-mail from the `rsMailingListEmails` recordset as its value. Change the value of this property to the following:

```
objCDO.To = rs_mailinglist_emails.Fields.Item("<EMAIL COLUMN>").Value
```

6. Go to Design view, highlight the ASP placeholder that represents the CDO code, and select Server Behaviors ➤ Repeat Region.

7. Select the rs_mailinglist_emails recordset from the menu and show all records. Click OK (Figure 5-23).

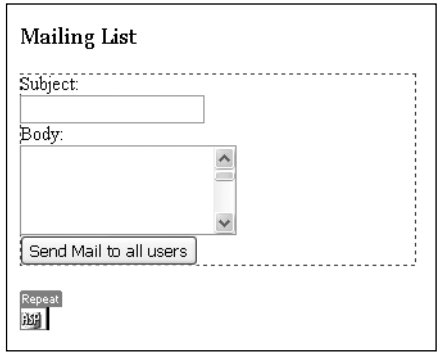


Figure 5-23. The mailing_list.asp page in the Dreamweaver document window

8. Finally, you need to make sure the ASP script along with the repeat region executes only when the form is submitted. In Code view, find this code at the beginning of the repeat region code block:

```
<%
While ((Repeat1__numRows <> 0) AND (NOT rs_mailinglist_emails.EOF))
%>
```

Add the following code *above* the start of the repeat region:

```
<%
If Request.Form("Submit") <> "" Then
%>
```

9. Then, in Code view, find the end of the repeat region code:

```
<%
  Repeat1__index=Repeat1__index+1
  Repeat1__numRows=Repeat1__numRows-1
  rs_mailinglist_emails.MoveNext()

Wend
%>
```

Add the following code directly *after* the end of the repeat region code:

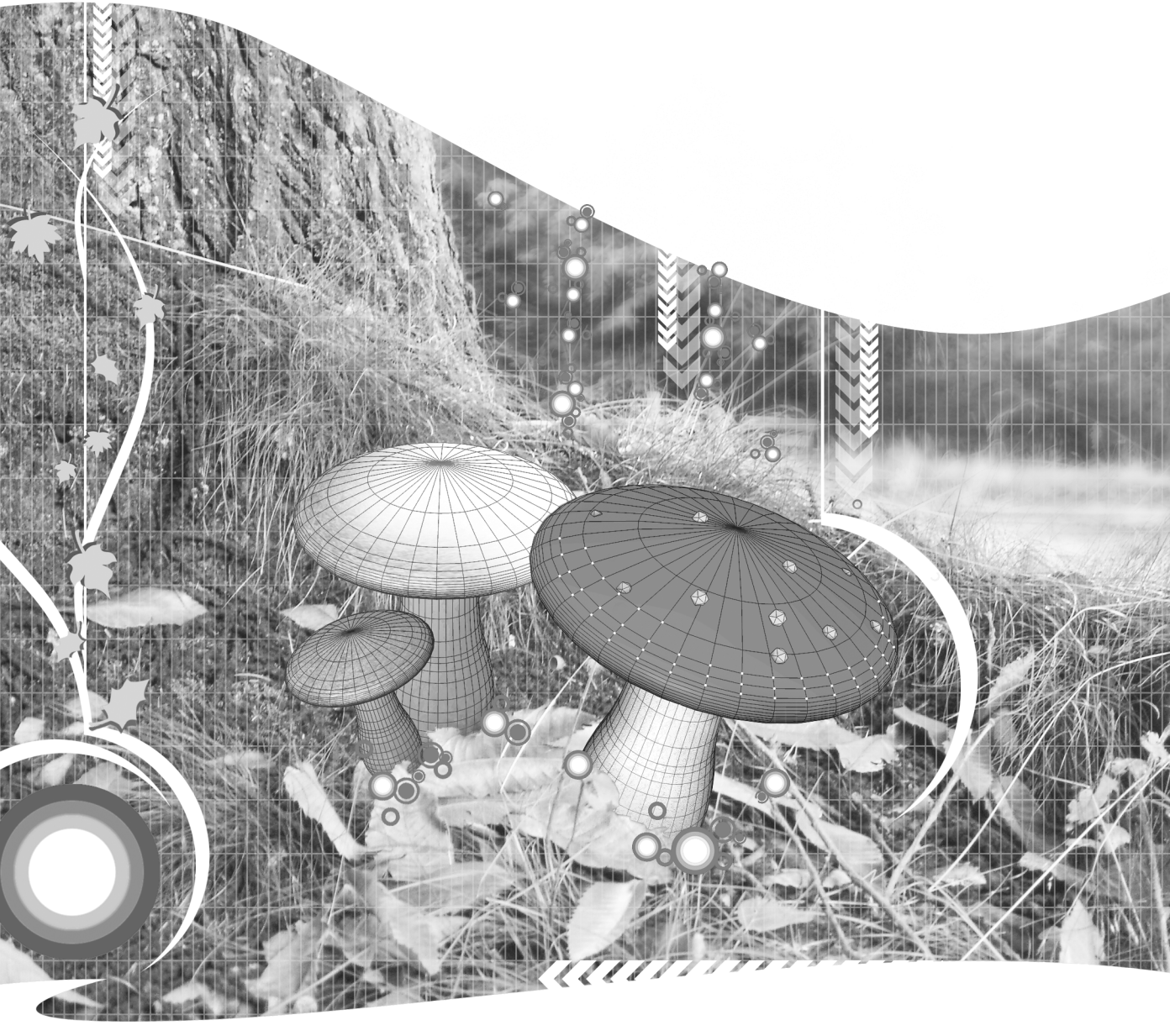
```
<%
Response.Redirect "mailing_list.asp?email=sent"
End If
%>
```

That's it. When the page is tested live, the recordset will retrieve all the e-mails from the `users` table. Once the form (Figure 5-24) is submitted, the ASP script will execute, repeatedly going through each e-mail address in the table and sending your message. Finally, it will redirect to the same page and add a URL parameter informing you that the e-mail was sent.

Figure 5-24. The mailing_list.asp page in the web browser before form submission. Input your own values and submit the form to test your mailing list application.

Conclusion

In this chapter, you learned how to send e-mails from within your own ASP pages. You looked at how to configure your SMTP server, and then you examined the syntax of three popular ASP mail components, CDO, AspEmail, and JMail, and how each can be used to send static e-mails with predefined values from an ASP page. Then you moved on to cover how to send dynamic e-mails based on conditions, how to change the values of the mail component properties on the fly, and even how to allow users to change those values. Finally, you walked through the creation of two full e-mail applications: one for sending users a forgotten password and the other for creating a mailing list. You should now be able to use the knowledge you gained here to create your own custom e-mail web applications for your website.



Chapter 6

BUILDING A RANDOM QUOTE GENERATOR

In this chapter, you'll build a database table to store quotes in, and then build a simple administration system to allow you to add, edit, and delete those quotes.

Your application will use Dreamweaver's Insert Record, Update Record, and Delete Record server behaviors to enable you to add, edit, and delete records in the database; it will also use the Repeat Region server behavior to help you display multiple records from the database on a page. If you need a little guidance on creating database tables, go back to Chapter 4, which covers this in depth. Otherwise, let's get going.

Creating the quotes database table

In Access, create a simple database table called tblQuotes, and use the following field names:

Field name	Data type	Settings
ID	Autonumber	Primary Key
CategoryID	Number	Default value = 0
AuthorID	Number	Default value = 0
Quote	Text	Size = 255

This very basic table will enable you to add your quotes to the database while catering for future expansion. In Chapter 7, you'll expand on this database to include categories and authors, but for now, you don't need to worry about them.

The same database table structure in SQL Server would be defined as follows:

Field name	Data type	Settings
ID	Int	Primary Key Identity = Yes Seed = 1 Increment = 1
CategoryID	Int	Allow Nulls = Checked Default value = 0
AuthorID	Int	Allow Nulls = Checked Default value = 0
Quote	Text	Allow Nulls = Unchecked Size = 255

In SQL Server, you would also need to give your SQL Server database user access rights to this table. You would need to grant select, insert, update, and delete permissions to the public role if your user was assigned to the public role in this database.

With your database structure in place, it's time to dive into Dreamweaver and begin building your administration system. If you're not yet familiar with creating connections to databases in Dreamweaver, please refer to the section titled "Making the connection" in Chapter 4.

Once you've created a connection to your database, you can then create the administration pages that will allow you to add, edit, and delete the data.

Building the form

To add data to your database, you'll make use of Dreamweaver's Insert Record server behavior. First, though, you'll build a simple form to enable you to add quotes to your database.

> *A quick word about forms and the ways in which you can submit their information between pages—a form has two methods of submitting its information:* `GET` *and* `POST`*. Using* `POST` *passes all a form's information to its destination (the form's* `ACTION` *property) in a way that doesn't allow the user to see it. Using* `GET` *passes the information through the querystring of the browser in name/value pairs. When information is passed through the browser's querystring, there's a much lower limit to the amount of information that can be passed from one page to the next.*

Create a new ASP VBScript page and save it to your site with the name `insert.asp` (Figure 6-1).

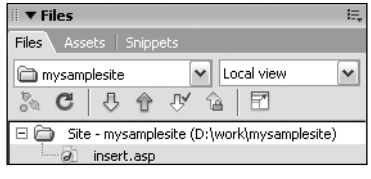


Figure 6-1. Saving insert.asp to your site

Once the page is saved to your site, you can begin working on it.

The first step is to add a form to the page. The form will contain a single form element into which you'll type your quotes, and a Submit button to submit the data to the database.

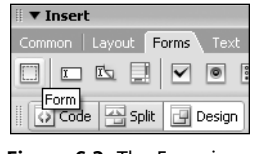


Figure 6-2. The Form icon on the Insert bar

1. From the Insert bar, select Forms and then click the Form icon to add a form tag to the page in Design view (Figure 6-2). A red dotted outline should appear in the layout area of the web page.

> *To avoid a bug in Dreamweaver, once the empty form has been added to the page, press the Enter key twice to insert a new paragraph (this will actually insert two sets of empty paragraph tags). Then press the Up arrow key to put the cursor back into the first paragraph before continuing.*

2. Next, click the Text Field icon next to the Form icon to begin adding a text field to the page. From a default installation of Dreamweaver, this will display the Input Tag Accessibility Attributes dialog box, which you can fill out as shown in Figure 6-3.

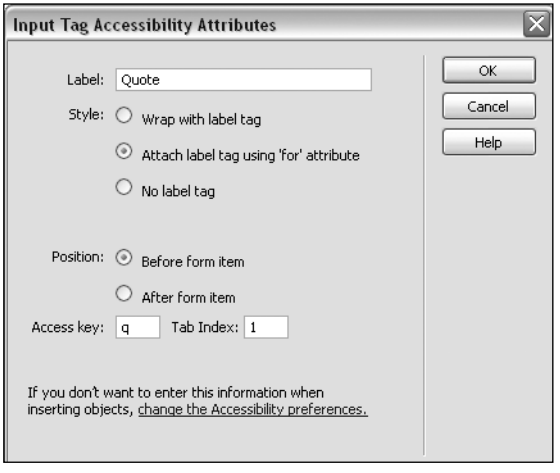


Figure 6-3. The Input Tag Accessibility Attributes dialog box

You've made use of one of Dreamweaver's accessibility features here. You don't have to do this—you can turn them off in Dreamweaver's Preferences *if you don't want to use these features, but it's advisable to make use of them whenever possible to make your websites accessible to more people. If you simply want to skip the accessibility dialog at this time, click* Cancel*, and your form element will be created normally. To turn these items off permanently, choose* Edit ➤ Preferences ➤ Accessibility*, and deselect the check boxes of those items you don't want to add accessibility attributes for.*

Next, you'll use the label attribute to label the text field accordingly. You'll also want to change the name given to this text field by Dreamweaver from textfield1 to something more meaningful, such as Quote.

3. Select the text field in Design view and change the name to Quote in the Property inspector. While you're there, set Char width to 50 and Max Chars to 255, as shown in Figure 6-4. This makes the text field display at 50 characters wide and allows a maximum of 255 characters to be entered. You're using 255 for the Max Chars value because that's the maximum field size you set in the database. If you try to insert more data than the table will allow, the operation will fail.

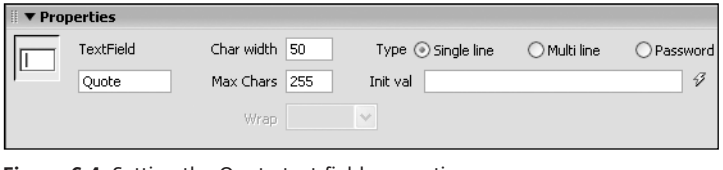


Figure 6-4. Setting the Quote text field properties

Changing the text field name in the Property inspector will at the same time change the ID that Dreamweaver assigned to the text field for you. Unfortunately, it won't change the `for` attribute of the label it inserted for you, so you need to manually change that to match the name you just gave the text field (Quote). To change the `for` attribute, go into Code view and type the correct name into the attributes value.

4. Now you need to add a Submit button to the form, which you do by clicking the Button icon on the Forms toolbar. Fill out the Input Tag Accessibility Attributes dialog box as before, giving the button a label of Submit, an access key of s, and a tab index of 2.

With that last step completed, your very basic form should look like Figure 6-5.

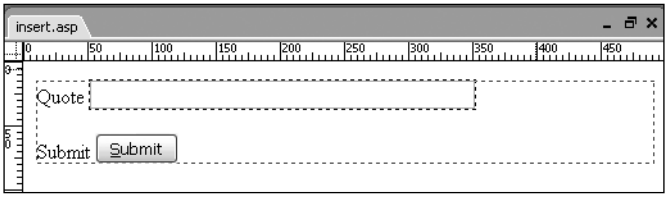


Figure 6-5. The basic Quote submission form

Now you're ready to add the Insert Record server behavior to the page to make this form work.

The Insert Record server behavior

To begin adding the Insert Record server behavior to the page, go to the Application panel and select Server Behaviors ➤ Insert Record. The Insert Record server behavior dialog box pops up for you to fill in.

If you haven't yet created a connection to your database, please see the "Making the connection" section in Chapter 4.

Select your connection in the Connection dialog box; when the tables list refreshes, select the table you're going to insert into. You'll be inserting a quote into the quotes table with this form, so select tblQuotes.

You can leave the After inserting, go to field empty. This will force the page to simply reload after an insertion has taken place. To redirect elsewhere after the insertion takes place, you would supply a page name here, along with any parameters that you may need. We'll return to this later in this chapter in the "Adding conditional code" section.

The rest of the dialog box should already be completed appropriately. The Get values from drop-down list should display form1, unless you happen to have renamed your form from the default. The Form elements box should be a single-item list, as you have only one form element on your form. The text entered into the Quote text field is therefore inserted into the Quote column as Text.

Your completed dialog box should look something like Figure 6-6.

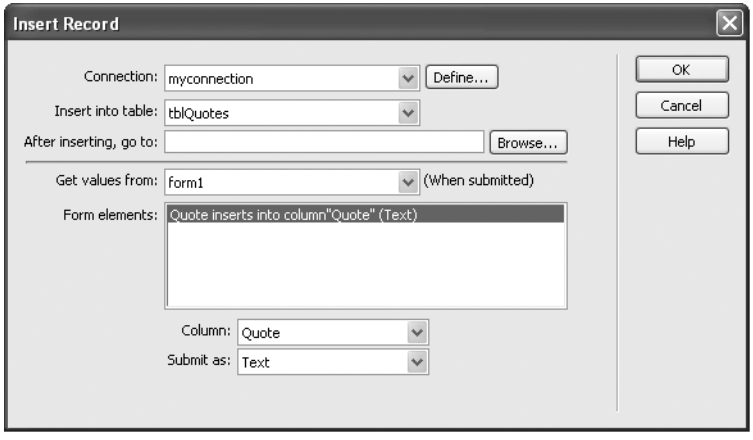


Figure 6-6. The completed Insert Record dialog box

Click OK to apply the Insert Record server behavior to the page and then save the page.

You can now perform a test of your new insert page by previewing the page (press F12 in Dreamweaver). When the page loads, enter a quote into the Quote field and then click Submit. The page will very quickly submit and reload. If you blink, you'll miss it and think nothing has happened—this is why it's always a good idea to add a little feedback to a form like this, so you know what's happening when you click buttons.

> *If you're stuck for ideas for quotes, check out* `www.brainyquote.com`*, which has some great ones.*

Adding conditional code

To provide feedback, you'll need to modify your Insert Record server behavior and then add a little code into the page to conditionally display some text.

First, double-click the Insert Record entry in the Server Behaviors panel to edit it. When the dialog box appears, enter the following code into the After inserting, go to field: `insert.asp?added=yes`.

`insert.asp` is the name of your insert page. With the preceding code, you're telling the server to return to this same page and also add the name/value parameter pair of added=yes. Your conditional code will check for the added parameter to see if its value is equal to yes, and display some text if it is.

Click OK to apply the changes to your page.

Switch views to Code view if you're in Design view, and place your cursor immediately to the left of the opening paragraph tag in the form. Switch back to Design view and press Enter to insert an empty paragraph tag above the Quote field and label.

In this new empty paragraph, enter the following text: Your quote has been added to the database. You'll now wrap this entire paragraph in some conditional code to make sure it displays only after an insert has taken place.

Switch into Code view and add the following code before the opening paragraph tag of the conditional text:

```
<%
  If Request.QueryString("added") = "yes" Then
%>
```

Then add the following code after the closing paragraph tag:

```
<%
  End If
%>
```

Your final piece of code for this segment of the page should look like the following:

```
<%
  If Request.QueryString("added") = "yes" Then
%>
<p>Your quote has been added to the database</p>
<%
  End If
%>
```

Save your page and preview it again. This time, when you test it (enter a quote and click the Submit button), the page will display the conditional text to let you know that the quote has been inserted into the database (see Figure 6-7).

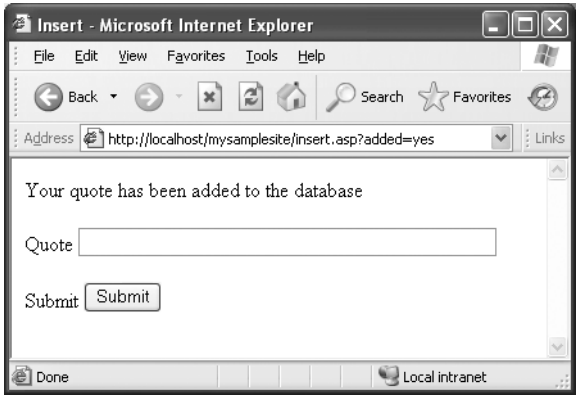


Figure 6-7. The conditional comment displayed after an insertion has taken place

You can now insert a quote into the database, but you still need to be able to edit the quotes that already exist in there, if there are any. To make it easier to do this, you need to create a page that

will display a list of all the quotes in the database. Each of these quotes should be a link that, when clicked, displays a form that allows you to edit that quote and save your changes back to the database.

To create this list of quotes, you'll create a new page and make use of the Repeat Region server behavior.

The Repeat Region server behavior

Create a new ASP VBScript page called `allquotes.asp` and save it to your site. In the Application panel, click Bindings and then click Recordset (Query) to launch the Simple Recordset builder.

> *For the purposes of this page, the Simple Recordset builder interface is sufficient. For any recordset you need to build that requires two or more parameters, you need to use the Advanced Recordset builder.*

Complete the Recordset dialog box as follows:

1. Name the recordset rsQuotes.
2. Select the connection to the database in the Connections drop-down list.
3. Select the quotes table—tblQuotes—(if it isn't already selected) in the tables list.
4. Check the Selected radio button, and then press Ctrl while you click the ID column and the Quote column in the list of available columns (pressing Ctrl allows you to select multiple columns at once).

   The Filter and Sort drop-down lists can be left set to None.

   The completed Recordset dialog box should look like Figure 6-8.

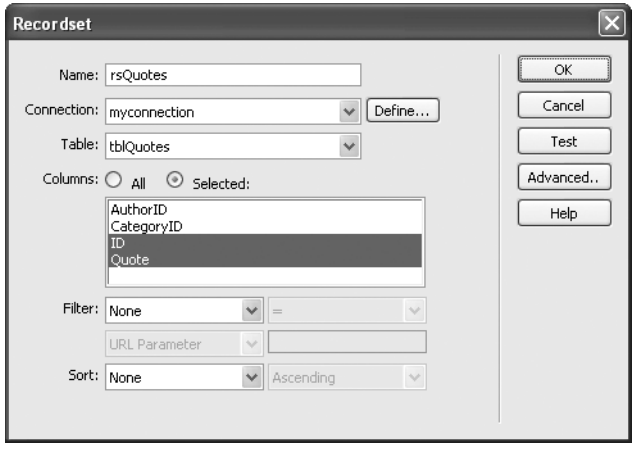


Figure 6-8. The completed Recordset dialog box

5. Click OK to create the recordset.

Now you can start adding the content to display on your page.

In Design view, enter a heading on the page, such as Quotes. Then press Enter to create a new paragraph underneath the heading.

Go to the Application panel and expand the rsQuotes recordset (Figure 6-9) so you can see all the available columns (all two of them).

Click the Quote entry to select it, and then click the Insert button at the bottom of the Bindings panel. This will add a data binding to your page; the Design view will reflect this by displaying a translator, or visual clue, for this binding. By default, this translator will look like this: {rsQuotes:Quote}, and will be displayed on a light blue background, to reflect the fact that you've bound the Quote column from the rsQuotes recordset to the page.

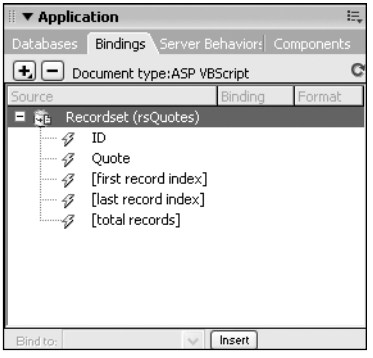


Figure 6-9. The rsQuotes recordset and its available columns

> *If you wish, you can change this setting in the* Preferences *so that only the curly braces are shown.*

Now to add the Repeat Region server behavior—first, you need to make sure that all the code you want to repeat on the page is selected. This means that you need to select the databound paragraph that you just created, including the paragraph tags. To ensure that this happens, select the data binding on the page (the translator text that you can see), and then click the <p> tag on the Tag Chooser at the bottom of the Design view window (see Figure 6-10).

Figure 6-10. The Tag Chooser with a paragraph tag selected

You can make doubly sure that the entire paragraph has been selected by switching into Code view and ensuring that the highlighted selection is correct. Switch back to Design view when you're happy!

In the Application panel, choose Server Behaviors ➤ Repeat Region. This will launch the Repeat Region server behavior dialog box. The rsQuotes recordset should be automatically selected for you. If it's not, select it in the list. Then set the Show option to All.

The completed dialog box should look like Figure 6-11.

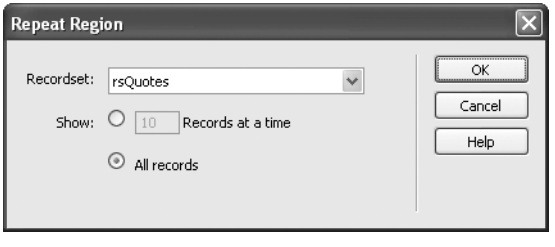


Figure 6-11. The completed Repeat Region dialog box

Click OK to apply the server behavior to the page. Your Design view will change to show an outline around the databound area with a tab above it labeled Repeat.

At this stage, you can preview your page to see the repeat region in action. If you have any quotes stored in your database already, they will all be displayed on this page now.

The next step is to make each quote into a link that goes to an edit page to allow you to edit the selected quote.

In Design view, click the databound element on the page {rsQuotes:Quote} and go to the Properties dialog box at the bottom of the screen. Click the folder icon to the right of the Link field (Figure 6-12) to launch the Select File dialog box.

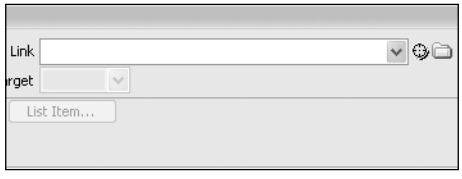


Figure 6-12. The folder icon to the far right of the Link field

The file you'll link to doesn't yet exist, so you can't simply click to select it. Instead, type the file name that you'll use to create your edit quote page (`editquote.asp`) in the File name field.

The `editquote.asp` page will allow you to edit a single quote. To do this, it will need to know which quote you want to edit. You tell `editquote.asp` this by passing it the ID of the quote you want to edit as a parameter. To set that parameter in your link, click the Parameters button next to the URL field.

In the Name column of the Parameters dialog box, enter the name ID and set its value to the ID column of your rsQuotes recordset by clicking the lightning bolt icon at the end of the field. (The lightning bolt icon is visible only when the value field has focus, so click inside it first). This will launch the Dynamic Data dialog box, in which you can select the ID column of the rsQuotes recordset (Figure 6-13).

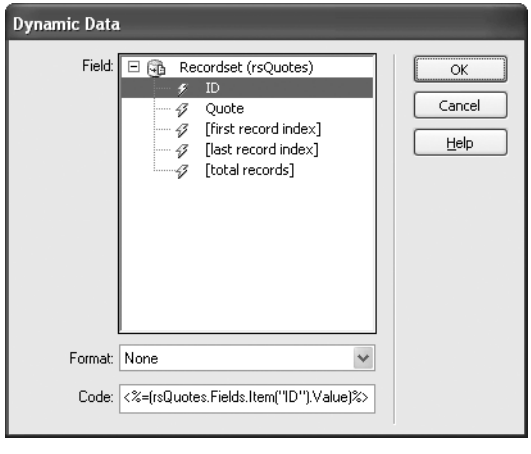


Figure 6-13. Selecting the ID column in the Dynamic Data dialog box

Click OK to return that selected value to the Parameters dialog box. The completed Parameters dialog box should look like Figure 6-14.

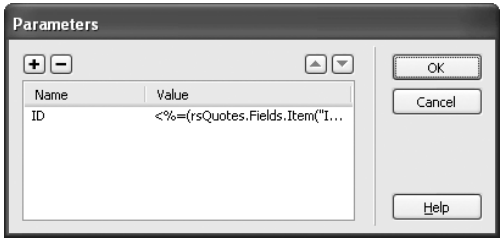


Figure 6-14. The completed Parameters dialog box

Click OK to return that data to the Select File dialog box that you initially began with. The completed URL, which you've been using Dreamweaver's dialog boxes to populate, will contain the following data:

```
editquote.asp?ID=<%=(rsQuotes.Fields.Item("ID").Value)%>
```

This is the link that has been created. Click OK to apply this code to the page. You can check that the code has been applied correctly by clicking the databound element on the page—{rsQuotes:Quote}—which is now a link, and checking the code in the Link element of the Properties dialog box.

Once you've saved your page, you can now preview it to see the full effect. Each of your quotes is now a link on the page; if you watch the status bar of your web browser while rolling over each of them in turn, you'll notice that the value of the ID parameter of each link is different.

Figure 6-15 shows how the completed page looks in a web browser.

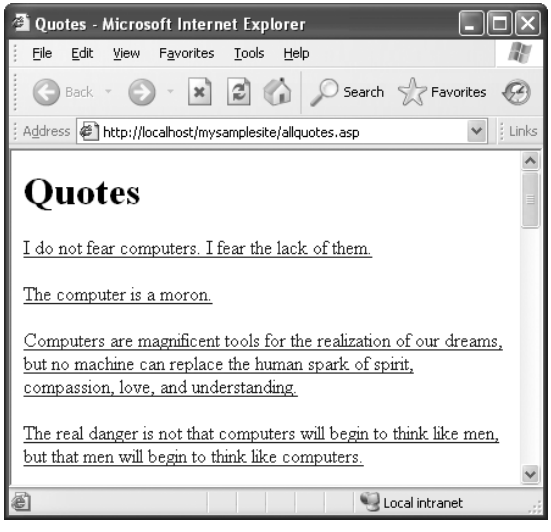

Quotes - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost/mysamplesite/allquotes.asp
Links
Quotes
I do not fear computers. I fear the lack of them.
The computer is a moron.
Computers are magnificent tools for the realization of our dreams, but no machine can replace the human spark of spirit, compassion, love, and understanding.
The real danger is not that computers will begin to think like men, but that men will begin to think like computers.
Local intranet

Figure 6-15. The completed Quotes page, so far . . .

With this page complete, you now need to create the page that all these links link to: `editquote.asp`.

The Update Record server behavior

The Update Record server behavior will enable you to update the contents of each quote in your database individually. You've already created a page that lists all the quotes in your database. Each one links to the `editquote.asp` page and passes it a unique ID that identifies a single record in the database to be updated.

Since `editquote.asp` needs to utilize the exact same form elements as the `insert.asp` page you created earlier, you'll use that page as a starting point. Open the `insert.asp` file and then use Save As to save it as a new page called `editquote.asp`.

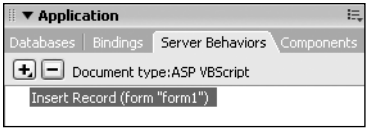


Figure 6-16. Selecting the Insert Record server behavior before removing it from the page

The first thing to do is to remove the Insert Record server behavior from this page. So, in the Application panel under the Server Behaviors tab, click the Insert Record entry, and then click the minus button above it to remove the entry from the page (as shown in Figure 6-16). This is the best way to remove a server behavior from the page, as it ensures that all the code relevant to that server behavior is removed. Manually deleting code may leave fragments of code on the page, which may cause problems.

In order to edit a record, you need to retrieve that record from the database. To do that, you'll build a recordset that will automatically return the record you want to edit. Your `allquotes.asp` page, which lists all the quotes in the database and links each one to this page, uses the ID of each quote as a parameter for you to uniquely identify it. You'll use this parameter when you build your recordset to retrieve that single record from the database.

You can launch the Recordset dialog box from the Server Behaviors panel (as well as the Bindings panel). To launch it from the Server Behaviors panel, choose Server Behaviors ➤ Recordset (Query).

1. In the Recordset dialog box, enter the name of your recordset as rsQuotes.
2. Select your database connection from the Connection drop-down list.
3. Select tblQuotes from the Table drop-down list.
4. Check the Selected radio button and highlight the ID and Quote columns in the Columns list.
5. Select ID in the Filter drop-down list, and leave = selected in the list next to it.
6. Select URL Parameter in the next list (it's selected by default).
7. Enter ID in the field next to that (it's entered by default).
8. Leave Sort as None—you're only retrieving a single record, so you don't need to sort anything!
9. Click OK to create the recordset.

Your completed Recordset dialog box should look like Figure 6-17.

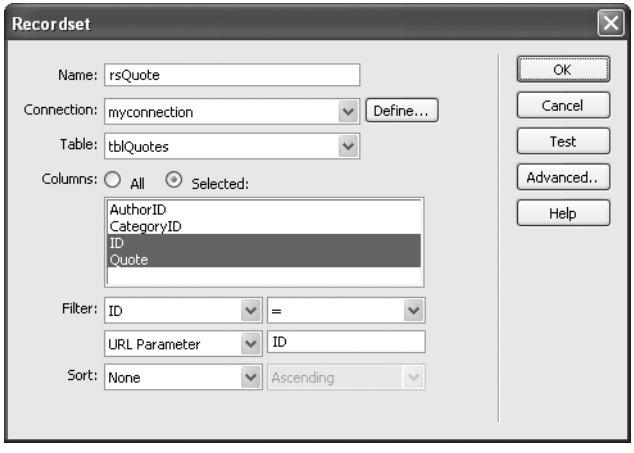


Figure 6-17. The completed Recordset dialog box

Click the Quote text field in Design view to select it, and then switch to the Bindings panel and expand the recordset so you can see the columns that have been retrieved in this recordset. Click to select the Quote column, and bind it to the Quote form element by clicking the Bind button at the bottom of the Bindings panel.

Now you need to add the Update Record server behavior to the page. Click Server Behaviors ➤ Update Record. The Update Record dialog box will appear.

1. Select your connection from the Connection drop-down list.
2. Select tblQuotes from the Table drop-down list.
3. Select rsQuotes from the Select record from drop-down list.
4. Select ID from the Unique key column drop-down list.
5. Enter editquote.asp?edited=yes in the After updating, go to field.
6. Make sure that form1 is selected in the Get values from field, and that the Quote form element is submitting to the Quote column as a Text data type.
7. When your dialog box is complete, click OK to apply the server behavior to the page.

Your completed Update Record dialog box should look like Figure 6-18.

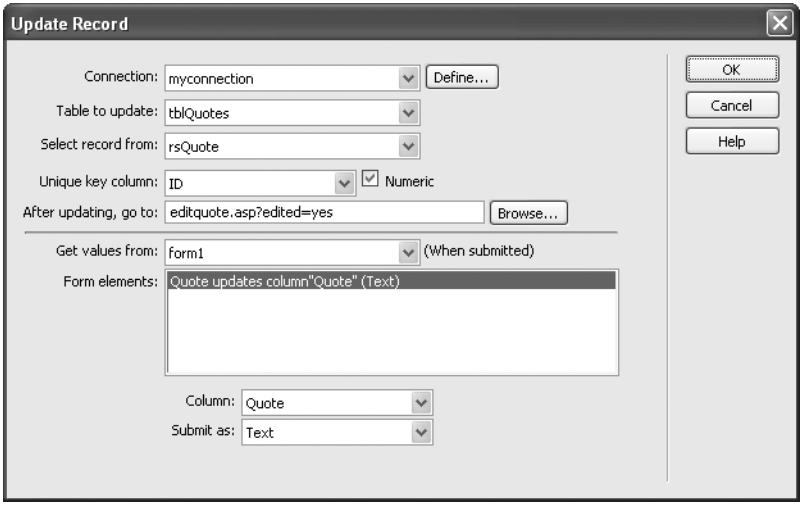


Figure 6-18. The completed Update Record server behavior dialog box

There are only a couple of small changes left to make, and this page will be complete. You need to change the conditional text to something more appropriate, and you also need to change the condition that it relies upon.

In Design view, change the text that reads Your quote has been added to the database to Your quote has been updated.

If you noticed in step 5 when filling out the Update Record server behavior, you entered the name of this page followed by the parameter name/value pair of edited=yes.

Switch to Code view and locate the following conditional statement:

```
If Request.QueryString("added") = "yes" Then
```

Change it to the following:

```
If Request.QueryString("edited") = "yes" Then
```

The final thing you need to do is add a link on this page that links back to the `allquotes.asp` page. In Design view, place the cursor after the Submit button and press Enter to start a new paragraph. Type Back to the quotes list as the link text. Then select the entire paragraph and make it a link in the Property inspector by clicking the Point to File icon (Figure 6-19) and dragging it to the `allquotes.asp` page in the Files panel. A line will stretch from the icon to the mouse while you click and drag; when you let go, the file you're pointing at will be the file you'll have created a link to. Check that `allquotes.asp` is entered in the Link field.

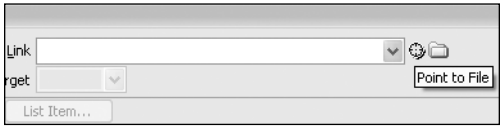


Figure 6-19. The Point to File icon, which looks like a bullseye

Save your page and test it now.

If you preview the editquote.asp page directly, it will always load and make the first quote from your database available for editing—this is a feature of the Simple Recordset builder. It has automatically added a default value for the ID parameter to the code for this page; if an ID value is not provided, it will use the default value of 1.

If you start your testing by previewing the allquotes.asp page, you can click a quote to edit it. You can then edit the selected quote and click Submit to save the changes. The edit page will refresh to tell you that the quote has been updated (see Figure 6-20). You can then click the link to go back to the allquotes.asp page and start again on another quote.

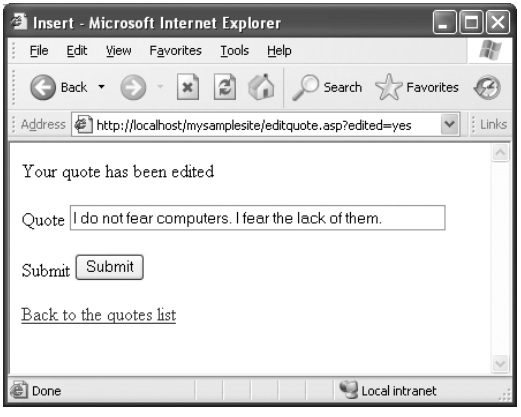

Insert - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/mysamplesite/editquote.asp?edited=yes
Links
Your quote has been edited
Quote I do not fear computers. I fear the lack of them.
Submit Submit
Back to the quotes list
Done
Local intranet

Figure 6-20. The updated Insert Quote page

Now you have a system that allows you to easily add and edit quotes in the database. What you need to complete this simple admin system is a facility to delete records that you no longer want in the database.

The Delete Record server behavior

For now, the last page you'll create in this simple admin system will be the delete.asp page. You'll also need to make another simple addition to the allquotes.asp page that will create a link to the delete.asp page and delete a selected record.

The Delete Record server behavior needs a recordset and a form on the page in order for it to be used. The easiest way to build the delete.asp page is to make a copy of the editquote.asp page and use that as a starting point, and then remove the bits on that page that you don't need.

1. Open editquote.asp and save it to your site as delete.asp.
2. Remove the Update Record server behavior by selecting it in the Server Behaviors panel and clicking the minus (-) button.
3. Highlight the conditional text in Design view, including the two ASP shields above and below the text, and press the Delete key to delete it.

With that done, you can start to add in the code you need to make this page delete quotes.

The page you're using as the starting point for this delete page, `editquote.asp`, already contains the recordset that will allow you to successfully run the delete command. It will automatically filter the database to a specific record based on the ID that will be passed from the `allquotes.asp` page, and display it to you in the form field.

Click Server Behaviors ➤ Delete Record to launch the Delete Record dialog box.

1. Select your connection from the Connection drop-down list.
2. Select tblQuotes from the Delete from table drop-down list.
3. Select the ID column in the Unique key column drop-down list, and ensure that Numeric is checked.
4. Select form1 (it's selected by default) in the Delete by submitting drop-down list.
5. Enter allquotes.asp in the After deleting, go to field.
6. Click OK to apply the server behavior to the page.

Your completed Delete Record dialog box should look like Figure 6-21.

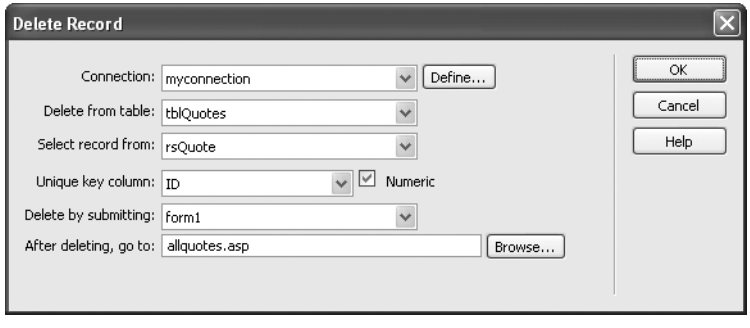


Figure 6-21. The completed Delete Record server behavior dialog box

Change the Submit button label—the text next to the button on the page in Design view—from Submit to Delete, and also change the value of the button itself. To do this, click the button to select it, and then go to the Property inspector and replace the value Submit with Delete.

Your `delete.asp` page is now completed, so save it to your site and close it for now.

When this page is accessed, the selected quote will be displayed in the Quote form element, and it will be deleted from the database if the Delete button is clicked. You left the link back to the `allquotes.asp` page there in case you get to the `delete.asp` page and decide not to delete the quote after all.

You now need to add another link to the `allquotes.asp` page that will link to the `delete.asp` page, and pass it the ID parameter so it knows which quote to delete.

Open the `allquotes.asp` page and place the cursor after the end of the databound value—{rsQuotes.Quote}—inside the repeat region in Design view. Enter a space and then type the word Delete. You'll now use the Property inspector to make this link to the `delete.asp` page and pass it the ID parameter with the correct value.

1. Highlight the word Delete, and in the Property inspector click the Browse for file icon.
2. In the Select File dialog box, click delete.asp in the list of files.
3. Click the Parameters button to launch the Parameters dialog box.
4. Click the plus (+) button to add a row to the list in the Parameters dialog box, and then enter ID in the Name column.
5. Click in the Value column, and then click the lightning bolt icon to bring up the Dynamic Data dialog box.
6. Click the ID field under the rsQuotes recordset to select it, and click OK.
7. Click OK to close the Parameters dialog box.
8. Click OK again to close the Select File dialog box. Your newly created dynamic link will now populate the Link field in the Property inspector (see Figure 6-22).

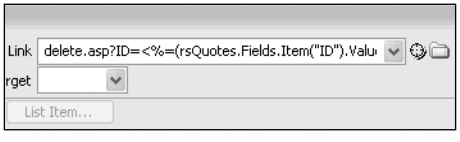


Figure 6-22. The completed Link field in the Property inspector

You can now save and preview this completed page (see Figure 6-23).

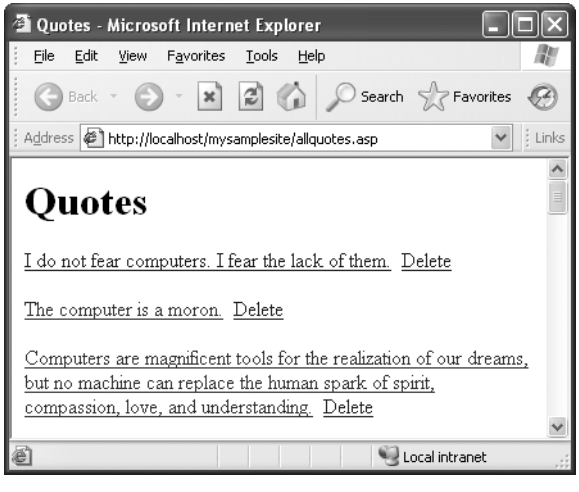

Quotes - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost/mysamplesite/allquotes.asp
Links
Quotes
I do not fear computers. I fear the lack of them. Delete
The computer is a moron. Delete
Computers are magnificent tools for the realization of our dreams, but no machine can replace the human spark of spirit, compassion, love, and understanding. Delete
Local intranet

Figure 6-23. The updated Quotes page, complete with Delete links

When you browse the `allquotes.asp` page, you'll see that each quote has a Delete link at the end of it, which, when clicked, will take you to the `delete.asp` page and offer you the chance to delete the selected quote (see Figure 6-24).

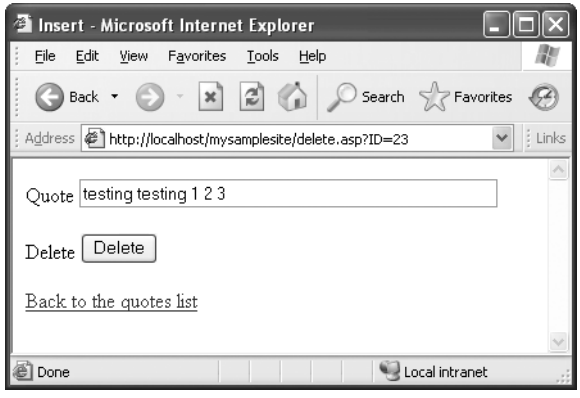
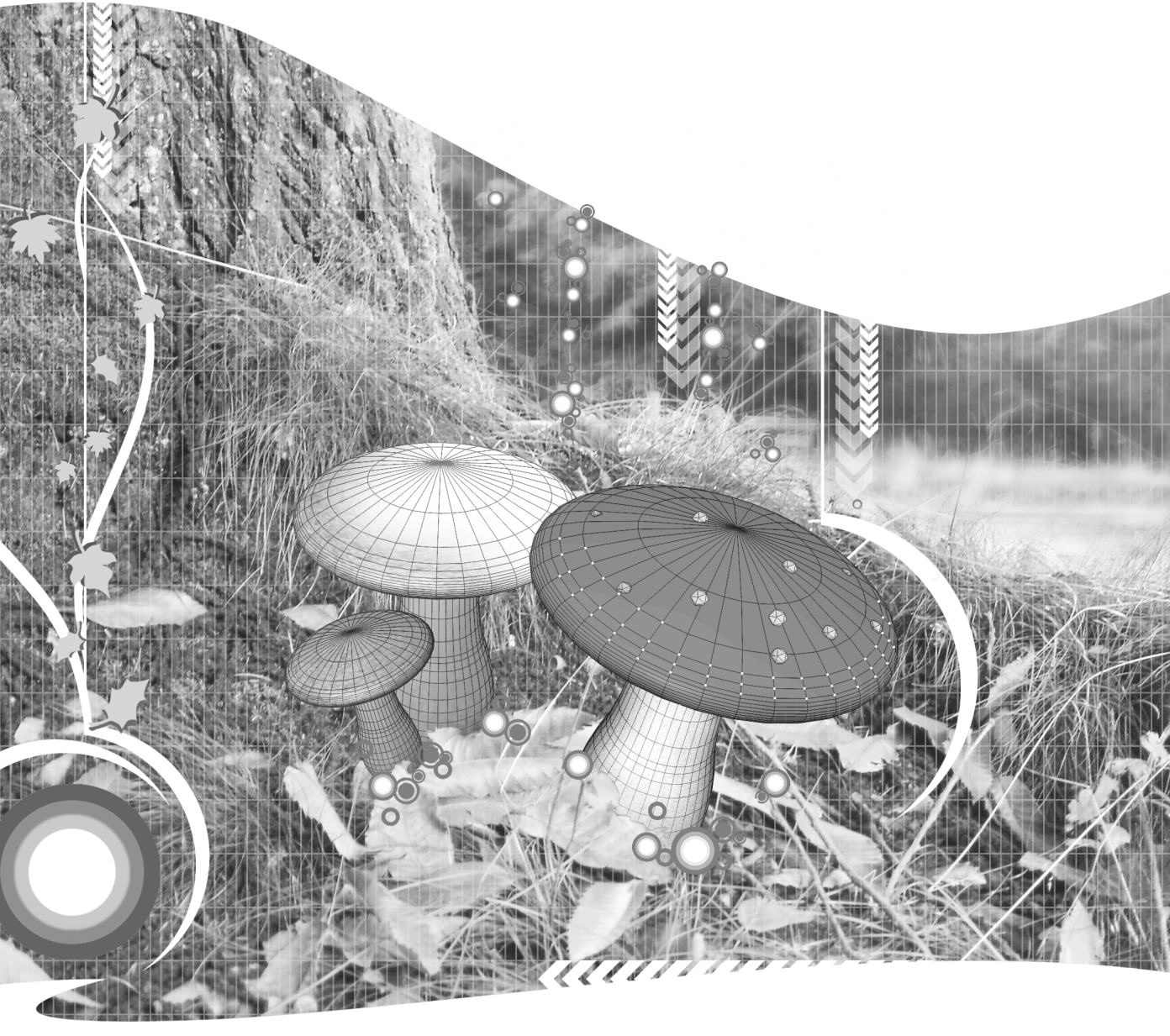


Figure 6-24. The Delete page

Conclusion

In this chapter, you've learned how to use Dreamweaver's Insert Record, Update Record, and Delete Record server behaviors by building a simple administration system that adds, edits, and deletes quotes from a single-table database. You've built a page to list all the quotes in the database, utilizing the Repeat Region server behavior, and you've created dynamic links to move from this page to edit and delete pages on which those relevant actions could take place.

In the next chapter, you'll extend this simple system to include multiple tables. This will allow you to utilize categories and authors for the database of quotes. You'll then move on to building a page that randomly retrieves and displays a quote from the database.



Chapter 7

COMPLETING THE QUOTES ADMINISTRATION SYSTEM

In the last chapter, you built a simple quotes administration system that used a single database table to store quotes and a set of ASP pages to administer that data through Add, Edit, and Delete actions, through the use of Insert Record, Update Record, and Delete Record server behaviors.

In this chapter you'll pick up where you left off in the previous chapter and add two more tables to the database to allow you to categorize the quotes and to see who actually said (or authored) them. You'll make use of **foreign keys** in the database, and you'll create a query that uses the `JOIN` keyword to gather together data from multiple tables into a single query. From that query, you'll retrieve a random quote and display it on a web page. You'll also update the administration pages we built in the previous chapter to incorporate categories and authors. All files can be downloaded from `www.friendsofed.com`.

Updating the quotes database table

First, to recap what you've done so far, open the existing Access database created in Chapter 6, and then open `tblQuotes`. The structure of `tblQuotes` is as follows:

Field name	Data type	Settings
ID	Autonumber	Primary Key
CategoryID	Number	Default value = 0
AuthorID	Number	Default value = 0
Quote	Text	Size = 255

This simple structure allows you to add quotes to the database, and it also has room to enable you to include categories and authors. These two columns, `CategoryID` and `AuthorID`, are foreign keys. They will reference the primary keys of two new tables that you're about to create.

The structure of the `tblQuotes` table in SQL Server is as follows:

Field name	Data type	Settings
ID	Int	Primary Key Identity = Yes Seed = 1 Increment = 1
CategoryID	Int	Allow Nulls = Checked Default value = 0
AuthorID	Int	Allow Nulls = Checked Default value = 0
Quote	Text	Allow Nulls = Unchecked Size = 255

Now you need to create two new tables: one to store the quote categories and another to store the quote authors. These are both very simple two-column tables. Create each table as described here and save it with the name shown (`tblCategories` and `tblAuthors`, respectively).

The tblCategories table structure is as follows:

Field name	Data type	Settings
CategoryID	Autonumber	Primary Key
Category	Text	Size = 255

The tblAuthors table structure is as follows:

Field name	Data type	Settings
AuthorID	Autonumber	Primary Key
Author	Text	Size = 255

Very simple!

Next, we need to create the relationship between these two tables and the main quotes table. In Access, you'll use the relationship management tool. You can launch this tool by clicking the Relationships icon, as shown in Figure 7-1.

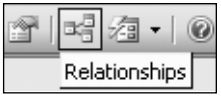


Figure 7-1. The Relationships icon in Access 2003

> *For a thorough grounding in how to create relationships in Access, please refer to the section "Creating a relationship in Access" in Chapter 4.*

This immediately displays the Show Table dialog box, from which you can choose the tables you want to create a relationship between. You're going to create a relationship between the main table, tblQuotes, and your other two tables, tblCategories and tblAuthors (Figure 7-2).

Figure 7-2. The Show Table dialog box

Select tblQuotes, tblCategories, and tblAuthors in the list by clicking each one while holding down the Ctrl key. Alternatively, with the top option selected, hold down the Shift key and select the bottom option in the list to select all items in between. Then click Add. You can also simply double-click each item in turn to add them to the relationships canvas. Once all three tables have been added to the canvas, click the Close button on the Show Table dialog to continue.

With the tables added to the canvas, click and drag the `AuthorID` column in `tblAuthors` to the `AuthorID` column in `tblQuotes`. When you release the mouse button, the Edit Relationships dialog appears (Figure 7-3). It shows the columns you want to relate to each other, and you can specify the relationship type that should be used.

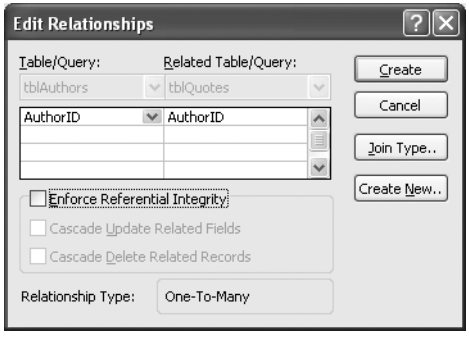


Figure 7-3. The Edit Relationships dialog box

By default, the Relationship Type in this dialog is One-To-Many, which is great, because that's the type of relationship you want to create. A many-to-many relationship type would require multiple related records in both tables, whereas one record in the `tblAuthors` table can be referenced by many individual records in the `tblQuotes` table using this one-to-many relationship type.

You also have the option of enforcing referential integrity on these database objects. To briefly recap the concept, if you tell the database that you want to use referential integrity, and you select the option to Cascade Delete Related Records, for example, then if you delete an author from the tblAuthors table, all of that author's quotes in the `tblQuotes` table will be deleted, too.

> *Referential integrity is covered in depth in Chapter 4 in the section "Relational databases and referential integrity."*

For this little project, you won't be using this feature, so simply click Create to create the one-to-many relationship and close the dialog.

Next, you'll create the same relationship between `tblCategories` and `tblQuotes`. Drag `CategoryID` from `tblCategories` and drop it over `CategoryID` in `tblQuotes`. Then click Create in the Edit Relationships dialog.

Your completed Relationships canvas should look like Figure 7-4.

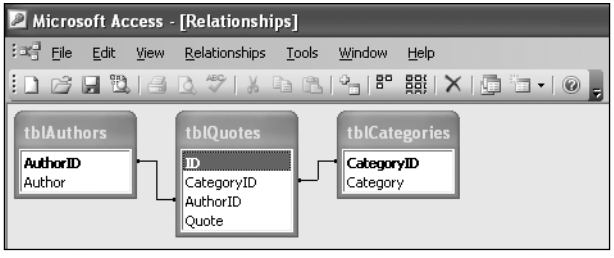


Figure 7-4. The completed Relationships dialog

> *We moved* `tblCategories` *to the right of* `tblQuotes` *to make it easier to see the relationship lines between all tables. This does not affect the relationships we created.*

You can now save and close the relationships editor window. Click File ➤ Save or click the Save icon (floppy disk) on the toolbar to save the relationships. Alternatively, you'll be prompted to save the relationships if you simply close the Relationships window.

With the database structure in place, it's time to dive into Dreamweaver and begin updating the administration system.

Author administration

To specify the author of each quote in the database, you need to have a list of authors. One way to achieve this is to create a system similar to the quotes administration system so that you could add, edit, and delete the author names in the database. However, to keep this system simple and get to the end result faster, you'll just build a single page that will allow you to add an author name to the database.

Building the Insert Author page

Open the `insert.asp` page created in the previous chapter and save it as `insertauthor.asp` by right-clicking the page tab and selecting Save As from the context menu that appears (Figure 7-5).

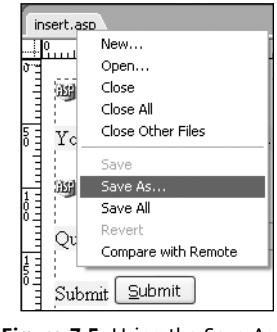


Figure 7-5. Using the Save As context menu option after right-clicking the page tab

Now you simply need to change anything that references "quote" to reference "author" instead.

First, change the text you can see in Design view from

Your quote has been added to the database

to

Your author has been added to the database

Also, change the label for the text field from Quote to Author. Click the text field to select it and then, in the Property inspector, change the name of the text field from Quote to Author. With the text field still selected, switch to Code view and change the `for` attribute of the `label` tag from `Quote` to `Author`. Dreamweaver should do this for us, but it doesn't!

The last change you need to make is to the Insert Record server behavior. Select Application ➤ Server Behaviors and you'll see that the Insert Record server behavior has a red exclamation point next to it (Figure 7-6). This is because it's looking for a `<form>` element on the web page called `Quote`, but we've changed the element's name to `Author`, so it can't find the element anymore!

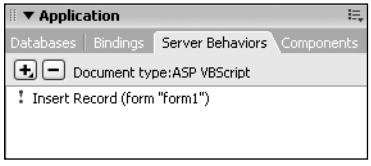


Figure 7-6. The exclamation point tells you that Dreamweaver has a problem with the code at the moment.

To remedy this situation, double-click the Insert Record server behavior to edit its properties. Click OK to get past the alert telling you that the server behavior can't find the form field called Quote.

Select tblAuthors from the Insert into table drop-down menu, and the dialog automatically updates itself to show that the Authors field will insert into the `Author` column as a data type of Text. Finally, change the After inserting, go to field so that it returns you to this `insertauthor.asp` page. Change `insert.asp` to `insertauthor.asp` and leave the parameters in place (Figure 7-7).

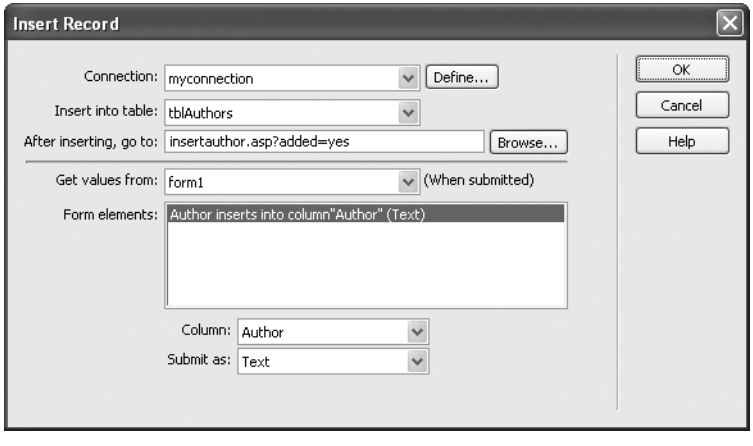


Figure 7-7. The updated Insert Record dialog

Click OK to apply the changes and then save the page.

If you now test the page in your web browser and submit a new author name to the database, the page will refresh to tell you that the new author was successfully added to the database (Figure 7-8).

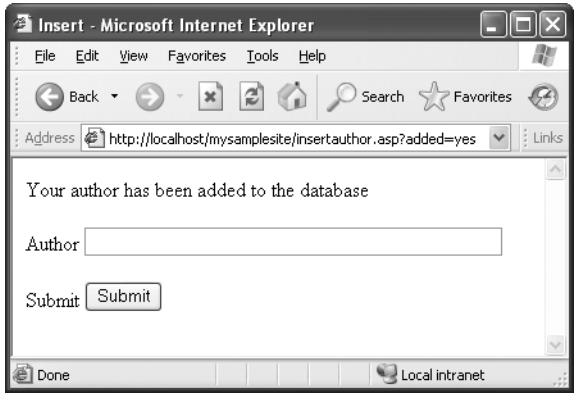


Figure 7-8. The working page after an author has been added to the database

> *For the sake of brevity, you won't produce a complete system to add, edit, and delete authors in this book. However, using the lessons from earlier in this chapter, you could quite easily do this yourself to better manage the data in the authors database table. Alternatively, you could download these edit and delete files from* `www.friendsofed.com`.

Next, you'll create the same page to allow you to add categories to the database.

Category administration

To specify the category of each quote in the database, you need to have a list of categories. As with the authors administration page, you'll just build a single page to allow you to add a category to the database.

Building the Insert Category page

Open the `insertauthor.asp` page just created and save it as `insertcategory.asp`. You'll need to make the same changes to this page that you just made to the Authors page.

First, change the text on the page in Design view from

Your author has been added to the database

to

Your category has been added to the database

Change the label for the text field from Author to Category. Click the text field to select it and then, in the Property inspector, change the name of the text field from Author to Category. Don't forget to switch to Code view and change the `for` attribute of the `label` tag from `Author` to `Category`, too.

Finally, you need to update the Insert Record server behavior. Click Application ➤ Server Behaviors and double-click the Insert Record entry in the list. Click OK to get past the alert telling you that the server behavior can't find the form field called Author.

Select tblCategories from the Insert into table drop-down menu. The dialog automatically updates itself to show that the Category field will insert into the Category column as a data type of Text. Finally, change the After inserting, go to field so that it returns you to this `insertcategory.asp` page. Change `insertauthor.asp` to `insertcategory.asp` and leave the parameters in place (Figure 7-9).

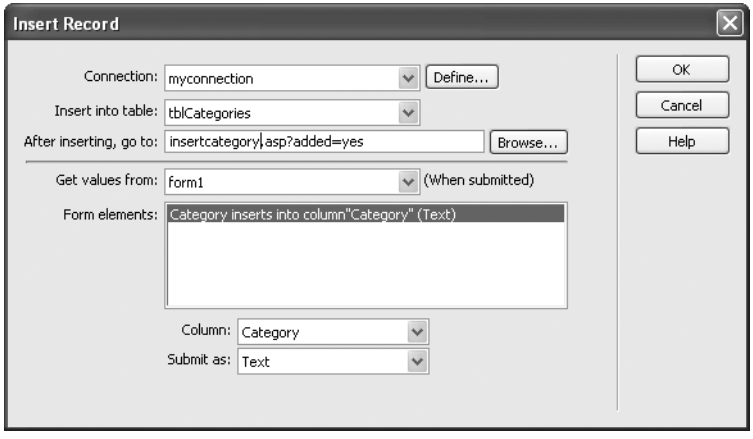


Figure 7-9. Updating the Insert Record dialog to insert a category

Click OK to apply the changes and then save the page.

If you now test the page in your web browser and submit a new category to the database, the page will refresh to tell you that the new category was successfully added to the database (Figure 7-10).

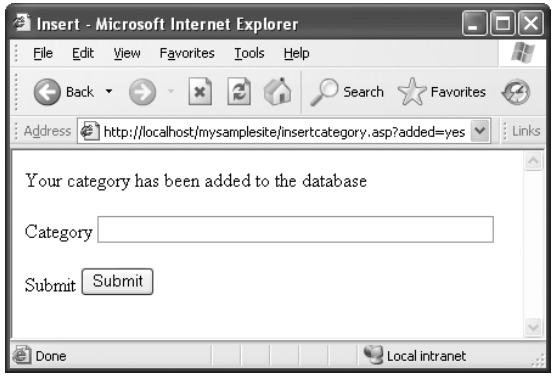


Figure 7-10. The Insert Category page after a category has been added to the database

Now you need to add these extra database fields, Author and Category, to the main quote administration page to make full use of them.

Updating the Insert Quote page

To specify each quote's author and category when you add quotes to the database, you need to amend the `insert.asp` page. You'll add two drop-down menus to the page, one for authors and one for categories, which will be populated by recordsets that pull data from the database.

Once the recordsets are created, you can bind them to select lists on the page and, finally, update the Insert server behavior to make use of the new `<form>` elements.

Creating the recordsets

Open the `insert.asp` page created previously. First, you'll create the two recordsets, `Authors` and `Categories`, that you need.

Building the Authors recordset

The `Authors` recordset will draw its data from the `tblAuthors` table and won't use any filters, as you want to include all the authors from the database. Here's how to create it:

1. In the Application panel, click Bindings ➤ Recordset (Query) to launch the Recordset builder dialog.
2. In the Recordset dialog, enter rsAuthors in the Name field, select your connection to the database in the Connection drop-down menu, and then make sure tblAuthors is selected in the Table drop-down menu (Figure 7-11). Make sure the All radio button next to Columns is selected—this is the default, so it should be selected already. Leave Filter set to None, as you don't want to filter out any records. You'll apply a Sort to the recordset to make it easy to locate an author when using the page later, so select Author from the Sort drop-down menu and leave Ascending selected as the sort order. This will retrieve all the authors in alphabetical order.

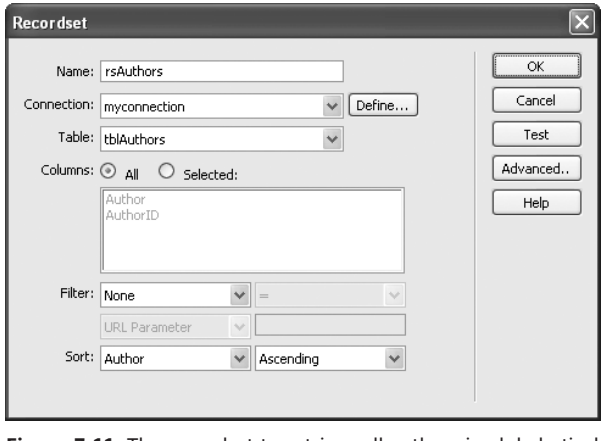


Figure 7-11. The recordset to retrieve all authors in alphabetical order

3. That's all you need to do for this recordset, so click OK to create it and close the Recordset dialog.

Building the Categories recordset

Your `insert.asp` page should now have the `rsAuthors` recordset listed in the Bindings panel. You just need to add the `Categories` recordset now, as follows:

1. Launch the Recordset dialog again by selecting Bindings ➤ Recordset (Query).
2. Enter rsCategories in the Name field, select your connection to the database in the Connection drop-down menu, and make sure tblCategories is selected in the Table drop-down menu (Figure 7-12). Again, you want to retrieve all columns, so leave the All radio button selected. Also, you want to retrieve the categories in alphabetical order for ease of use when using the page later, so select Category in the Sort drop-down menu and leave Ascending selected as the sort order.

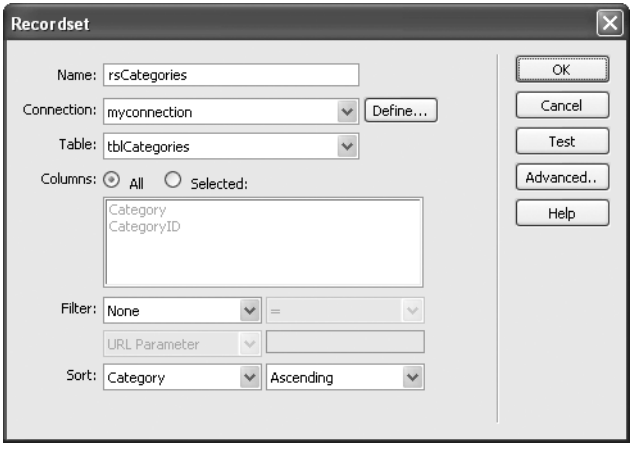


Figure 7-12. The rsCategories recordset being built

3. Click OK to create the recordset and close the Recordset dialog.

Building dynamic select lists

Click in Design view to place the cursor after the closing ASP shield of the conditional code, just before the label for the quote `<form>` element, and press Enter to insert a new paragraph above that quote `<form>` element.

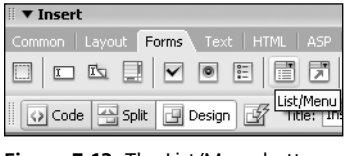


Figure 7-13. The List/Menu button on the Insert bar in the Forms category

In this new paragraph, insert a new list/menu `<form>` element either by clicking the List/Menu icon in the Insert bar (Figure 7-13) or by selecting Insert ➤ Form ➤ List/Menu. This will add a blank drop-down menu to your Design view page. In Code view, this is defined as a **select list**.

The Input Tag Accessibility Attributes dialog will open, unless you have turned it off in your preferences—but you didn't do that, did you?

Enter Author as the Label for this list/menu item and choose to Attach label tag using 'for' attribute. Enter a in the Access key field and 1 in the Tab Index field, and then click OK to add this `<form>` element to

the page (Figure 7-14). You will need to go through the <form> elements in Code view to change each of the element's respective Tab Index values so that they are sequential. This will allow you to use the Tab key to navigate this form; each press of the Tab key will move you to the next numbered form element in ascending order.

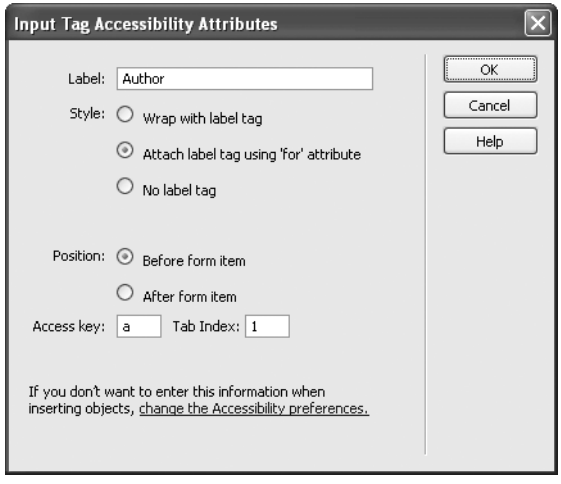


Figure 7-14. Specifying how this list/menu item should be added to the page in the Input Tag Accessibility Attributes dialog

Although Dreamweaver's simplistic naming conventions will make this code work, we suggest giving <form> *elements* name*s and* id*s that make sense to you, so that when you return to the code later on (perhaps to debug a problem), you can immediately make sense of all the elements. When you insert the list/menu item, Dreamweaver gives it an* id *that starts with* select *and, if it's not the first of its kind on the page, appends a sequential number to it.*

The new list/menu item was just given a name and id attribute of select; the for attribute of the <label> tag was given the same value. Switch into Code view and change these three items to Author. Your updated code should look like this:

```
<label for="Author">Author</label>
<select name="Author" accesskey="a" tabindex="1" id="Author">
</select>
```

Now switch back to Design view, click the canvas after the Author drop-down menu, and press Enter to insert another new paragraph between the Author drop-down menu and the quote text field.

In this new paragraph, insert another new list/menu <form> element by selecting Insert ➤ Forms ➤ List/Menu. In the Input Tag Accessibility Attributes dialog, enter Category as the Label and choose to Attach label tag using 'for' attribute. Enter c in the Access key field and a 2 in the Tab Index field, and then click OK to add this <form> element to the page.

The new list/menu item was given a `name` and `id` of `select`; the `for` attribute of the `label` tag was given the same value. Switch into Code view and change these three items to `Category`. Your code should look like this:

```
<label for="Category">Category</label>
<select name="Category" accesskey="c" tabindex="2" id="Category">
</select>
```

With the select lists added to the page, you can now bind the recordset data to them, as follows:

1. In Design view, click the Author select list to select it.
2. In the Property inspector, click the Dynamic button (see Figure 7-15).
3. In the Dynamic List/Menu dialog that appears, the author `<form>` element will already be selected in the Menu drop-down. If it is not for some reason, make sure you select it before continuing.
4. The next section of the dialog allows you to specify Static options, or items that will appear in the Author drop-down menu that do not come from the database. You'll add a Please select static option, as it is often seen in drop-down menus like this. Click the + button to add a row to the grid of Static options. You don't need a value associated with this static option, so click in the Value column to highlight the word "value" and then delete it. Click in the Label column and type Please select. Clicking anywhere in the dialog (that isn't a button) or pressing Enter will apply your changes to the Static options list without closing the dialog. The Static options row you just entered will turn blue to show that it is saved in the list.
5. Select the rsAuthors recordset from the Options from recordset drop-down menu, choose AuthorID from the Values drop-down menu, and choose Author from the Labels drop-down menu (Figure 7-16).

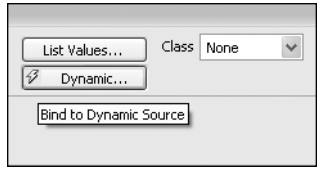


Figure 7-15. The Dynamic button in the Property inspector when a list/menu is selected

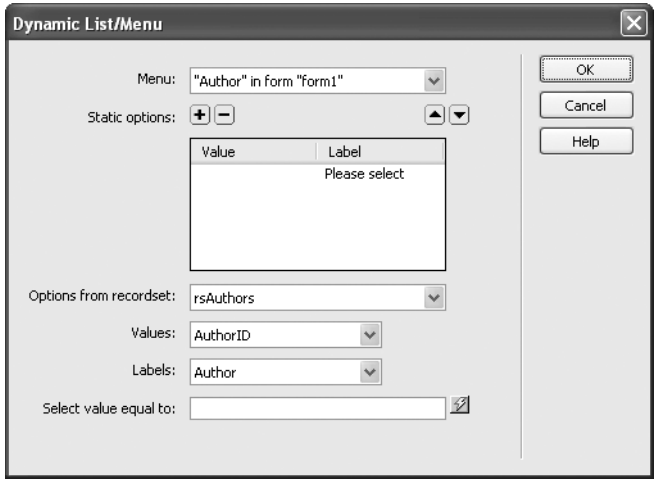


Figure 7-16. Building a data-driven Author drop-down menu

6. You don't need to specify a value to select in the list on this page—you'll do that later on the `editquote.asp` page, so click OK to apply the code to the page.

You need to follow the same steps for the Category drop-down menu, except that you'll use information relevant to categories instead of authors:

1. In Design view, click the Category drop-down menu to select it.
2. In the Property inspector, click the Dynamic button.
3. The category `<form>` element should be selected in the Menu field. If it isn't, select it now.
4. Add a Please select static option to the dialog as outlined in step 4 of the preceding instructions for creating the Authors drop-down menu.
5. From the Options from recordset drop-down, select rsCategories, and then select CategoryID from the Values list and Category from the Labels list (Figure 7-17).

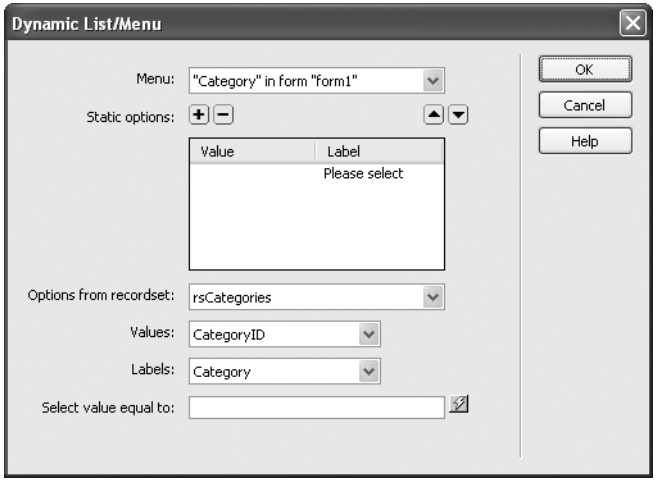


Figure 7-17. Building a data-driven Category drop-down menu

6. Again, you don't need to specify a value to select in the list, so click OK to close the Dynamic List/Menu dialog and add the dynamic drop-down code to your page.

Updating the Insert server behavior

Now that your dynamic drop-down menus are in place, all you need to do to complete this page is update the Insert server behavior to include these two new items: Author and Category.

1. In the Application panel, click Server Behaviors and then double-click the Insert Record server behavior to begin editing it.
2. In the Form elements section of the dialog, click Author to select it if it is not already selected by default. Then choose AuthorID from the Column drop-down menu. Leave Numeric selected in the Submit as drop-down menu.

3. In the Form elements section again, click Category to select it if it is not already selected by default. Then choose CategoryID from the Column drop-down menu and leave Numeric selected in the Submit as drop-down menu (Figure 7-18).

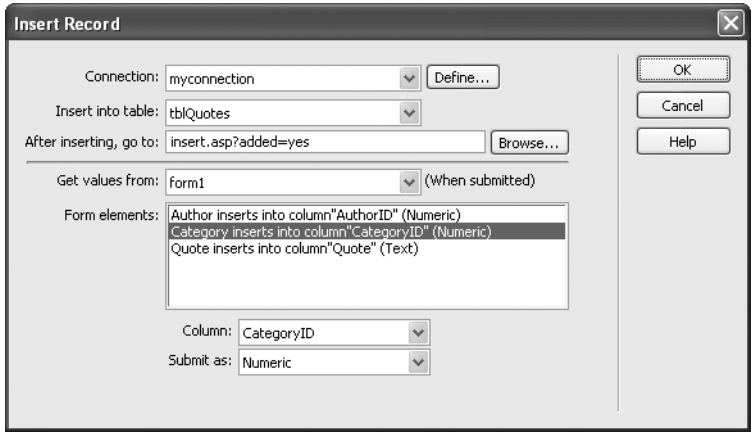


Figure 7-18. The updated Insert Record dialog to add an author and a category, along with the quote itself

4. Click OK to apply the changes to the page.

This page is now complete, so save your page but leave it open in Dreamweaver for the moment. Next, you need to make similar changes to the `editquote.asp` page that you just made to this `insert.asp` page.

Updating the edit page

So far, the `editquote.asp` page allows you to only edit the actual quote text and nothing else. Now that you have a system to add authors and categories into the database, and to assign an author and a category to each quote that you add, and the database has been set up to relate those records from the authors table and from the categories table to the quotes table, you can make full use of the extra data in the database and complete the administration system.

In this section, you'll add two dynamic drop-down menus to the `editquote.asp` page in the same way you did for the `insert.asp` page, but on the `editquote.asp` page you'll make them automatically select the correct current author and category assigned to the quote, if any.

Copying recordsets

You can save yourself some time by copying work you've already done instead of repeating the same steps each time:

1. Open the `editquote.asp` page so that it's ready to accept the recordsets you're about to copy from the `insert.asp` page.
2. If you closed the `insert.asp` page previously, open it now; otherwise, click the `insert.asp` page tab in Design view (Figure 7-19).

Figure 7-19. The page links, with the editquote.asp tab currently in focus

3. Select Application ➤ Bindings to show the recordsets, and then click rsAuthors to select it. You can either right-click the recordset and then choose Copy from the context menu (Figure 7-20), or you can use the standard Windows keyboard shortcut of Ctrl+C to copy it.

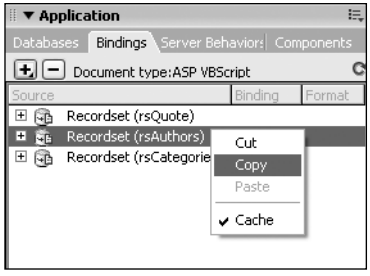


Figure 7-20. The context menu after a recordset is right-clicked in the Bindings panel

4. You have a few ways to paste this recordset into the Bindings panel of the `editquote.asp` page, including the keyboard shortcut method of pressing Ctrl+V, or by right-clicking in the white area of the Bindings panel and selecting the Paste option. A third option is clicking the menu icon in the blue Application panel bar and choosing Paste from that menu (Figure 7-21).

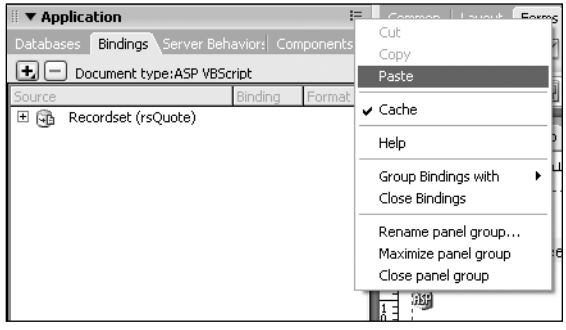


Figure 7-21. Pasting the recordset using the panel group menu option (instead of the right-click context menu)

5. Click the editquote.asp page tab to focus that page, and then paste the `rsAuthors` recordset using your preferred method.
6. Switch back to `insert.asp`, copy the `rsCategories` recordset, and then paste that recordset into `editquote.asp`.

Adding authors and categories to the edit page

Adding dynamic drop-down menus to a page in Dreamweaver is quite easy. Now that you've copied the recordsets from `insert.asp` to retrieve the list data from the database, all you need to do is use one of Dreamweaver's built-in wizards to set the list to behave the way you want.

You'll add dynamic drop-downs in much the same way you added the list/menu items for Author and Category to the `insert.asp` page:

1. Place the cursor after the closing ASP shield of the conditional code, just before the label for the quote `<form>` element, and press Enter to insert a new paragraph above it.
2. In this new paragraph, insert a new list/menu `<form>` element. When the Input Tag Accessibility Attributes dialog opens, enter Author as the Label and choose to Attach label tag using 'for' attribute. Enter a in the Access key field and a 1 in the Tab Index field, and then click OK to add this `<form>` element to the page.
3. Once again, your new list/menu item is given a `name` and an `id` attribute of `select`; the `for` attribute of the `<label>` tag was given the same value. Switch into Code view and change these three items to `Author`. Your updated code should look like this:

```
<label for="Author">Author</label>
<select name="Author" accesskey="a" tabindex="1" id="Author">
</select>
```

4. Switch back to Design view and click the canvas after the Author drop-down menu. Press Enter to insert another new paragraph between the Author drop-down menu and the quote text field.
5. Insert another new list/menu `<form>` element, and in the Input Tag Accessibility Attributes dialog, enter Category as the Label and choose to Attach label tag using 'for' attribute. Enter c in the Access key field and 2 in the Tab Index field, and then click OK to add this `<form>` element to the page.
6. Again, your new list/menu was given a `name` and an `id` attribute of `select`; the `for` attribute of the `<label>` tag was given the same value. Switch to Code view and change these three items to `Category`. Your code should look like this:

```
<label for="Category">Category</label>
<select name="Category" accesskey="c" tabindex="2" id="Category">
</select>
```

> *Don't forget to update the other form elements'* `tabindex` *values so that they are all sequential and none are repeated. Otherwise they won't work as you expect!*

Next, you'll move on to the dynamic data side of things on this page.

Updating the Quotes recordset

You need to make one small but significant change to the `rsQuotes` recordset. You now need to retrieve the extra columns for `AuthorID` and `CategoryID` to be able to modify that data on your form.

1. Double-click the rsQuotes recordset in the Bindings panel, and it will open up in the Recordset dialog.
2. You need to choose to retrieve All columns instead of Selected as you had previously (Figure 7-22). Everything else on this dialog can stay unchanged.

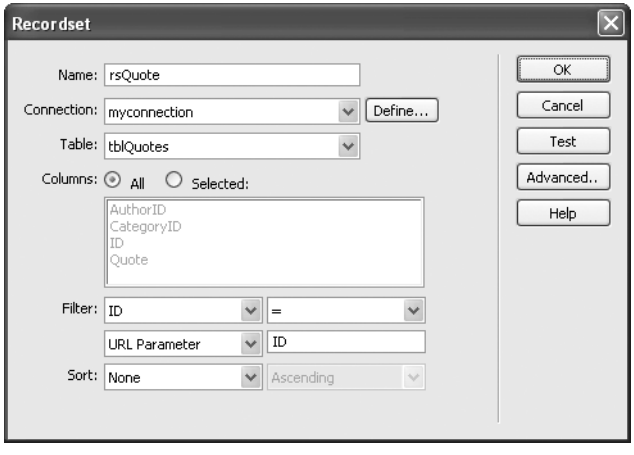


Figure 7-22. Updating the rsQuotes recordset to retrieve only a specific quote based on the URL parameter

3. Click OK to save those changes to the recordset, and then save your page.

Binding dynamic select lists with a selected item

You now have all the elements in place on this page. All that's left to do is to bind the data to the drop-down menus and make them work.

1. In Design view, click the Author drop-down menu to select it.
2. In the Property inspector, click the Dynamic button.
3. In the Dynamic List/Menu dialog that appears, the author `<form>` element will already be selected in the Menu drop-down menu. If it isn't selected for some reason, make sure you select it before continuing.
4. You'll add a Please select static option to the list as per the `insert.asp` page. Click the + button to add a row to the grid of Static options. Click in the Value column to highlight the word "value" and delete it. Then click in the Label column and type Please select. Next, select the rsAuthors recordset in the Options from recordset drop-down menu, choose AuthorID in the Values drop-down menu, and choose Author in the Labels drop-down menu.

You want this list to automatically select the value stored in the database for the quote being edited, if such a value exists. To do this, you need to tell the list to look out for a specific value when it populates the Author drop-down menu values from the `rsAuthors` recordset and, if it finds the value you specify, mark it as selected in the list. Of course, you need to tell it to look out for a dynamic value—one that will change each time—so you can't simply type a static value into the field. Instead, you'll specify a value from the `rsQuotes` recordset.

5. To do this, click the lightning bolt icon at the end of the Select value equal to field to launch the Dynamic Data dialog.
6. Expand the `rsQuotes` recordset and click AuthorID to select that column (Figure 7-23).
7. You will see the relevant code written automatically in the Code field at the bottom of the dialog.
8. Click OK to close this dialog and apply this dynamic code to the previous dialog. The code that was generated for you will now appear in the Select value equal to field.

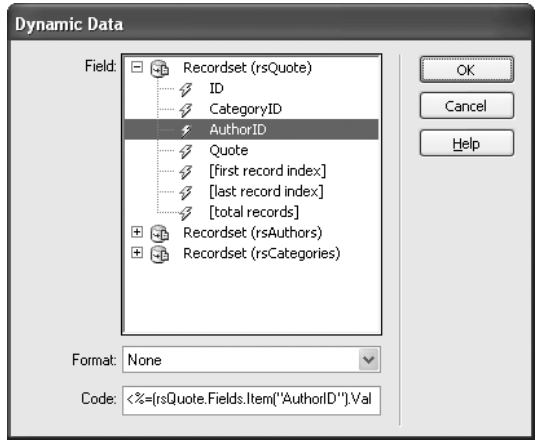


Figure 7-23. Selecting which column holds the AuthorID value that will be marked as selected in the drop-down menu

This Dynamic List/Menu dialog is now complete and should look like Figure 7-24.

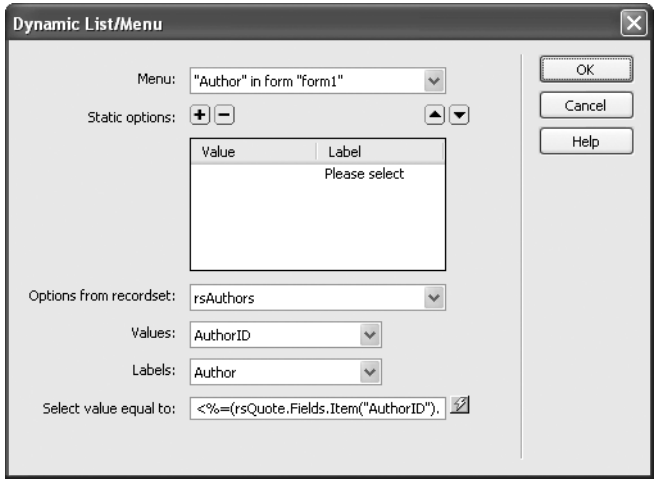


Figure 7-24. Adding the code required to mark the correct author as selected in the Author drop-down menu

Click OK to apply the code to the page, then save the page.

You need to follow the same steps for the Category drop-down menu, except that you will use information relevant to categories instead of authors:

1. In Design view, click the Category drop-down menu to select it.
2. In the Property inspector, click the Dynamic button.
3. The category form element should be selected in the Menu field. If it isn't, select it now.
4. Add a Please select static option to the dialog as outlined in step 4 of the previous set of instructions. From the Options from recordset drop-down, select rsCategories, and then select CategoryID from the Values list and Category from the Labels list.
5. Click the lightning bolt icon next to the Select value equal to field and choose the CategoryID column from the rsQuotes recordset in the Dynamic Data dialog that appears.
6. Click OK to close the Dynamic Data dialog, and then click OK to close the Dynamic List/Menu dialog and add the dynamic list code to your page.

Updating the Update server behavior

With your dynamic drop-down menus in place, all you need to do to complete this page is update the Update server behavior to include these two new items, Author and Category:

1. In the Application panel, click Server Behaviors and then double-click the Update Record server behavior to begin editing it.
2. In the Form elements section of the dialog, click Author to select it if it isn't already selected by default. Then choose AuthorID from the Column drop-down menu and leave Numeric selected in the Submit as drop-down menu.
3. In the Form elements section again, click Category to select it if it isn't already selected by default. Then choose CategoryID from the Column drop-down menu and leave Numeric selected in the Submit as drop-down menu (Figure 7-25).

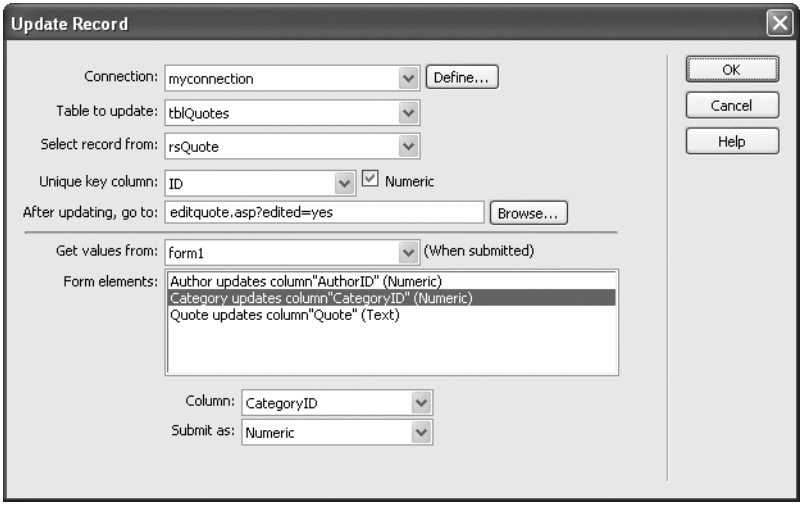


Figure 7-25. The updated Update Record dialog. Now authors and categories can be updated!

4. Click OK to apply the changes to the page, and then save your page.

With the `editquote.asp` page now completed, you can move on to build the page that will retrieve a random quote from the database to round off this little application.

If you haven't added any quotes to your database yet, now would be a great time to do so before moving on to the next step of retrieving a random quote. You can download data from `www.friendsofed.com` if you need to.

The random quote generator

With the administration system complete, you can now build a simple page to randomly select a quote from the database and display it, along with the author of the quote and the category it has been classified under. To help you achieve that goal, you'll build a query in the database to gather the data from the three tables into one place from which you can select your data.

Creating a join in the database

Creating the join in Access can be done using a wizard or manually in a visual designer. We suggest using the visual designer method over the wizard, as follows:

1. Open your Access database and click Queries in the left menu to display any queries you may already have in there.
2. Double-click the Create query in design view list entry and the query designer window will open. The first thing you need to do is to add the tables that will be needed to the design of this query. You can always add more later, but you're adding only three here, so you may as well add them all up front.
3. Select tblAuthors, tblCategories, and tblQuotes in the Show Tables dialog (Figure 7-26) and click Add to add them to the query designer window.

Figure 7-26. Adding tables to the design of the query

4. Click Close to close the Show Tables dialog.

The query designer window displays the three tables with a black line joining tblAuthors and tblCategories to tblQuotes (Figure 7-27). This window shows you the relationships among the three objects you created earlier in this chapter.

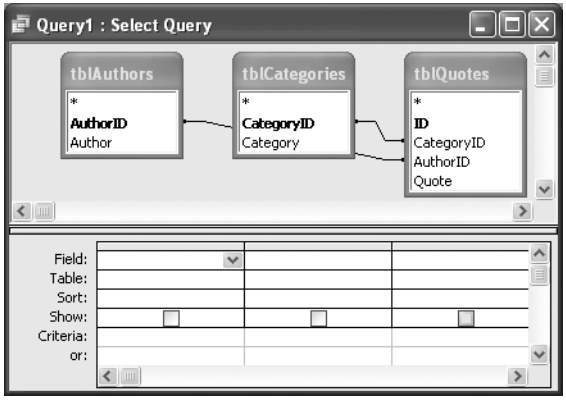


Figure 7-27. Query designer window showing the tables and their relationships

Now you can begin adding the columns to this query that you want to select from all three tables. To add a column to the query SELECT statement, you just need to double-click it.

5. In the tblQuotes table, double-click the Quote column. It will appear in the rows in lower window of the query designer. Next, double-click the Author field in tblAuthors to add that to the selected items. Finally, double-click the Category field in tblCategories. The query designer should now look like Figure 7-28.

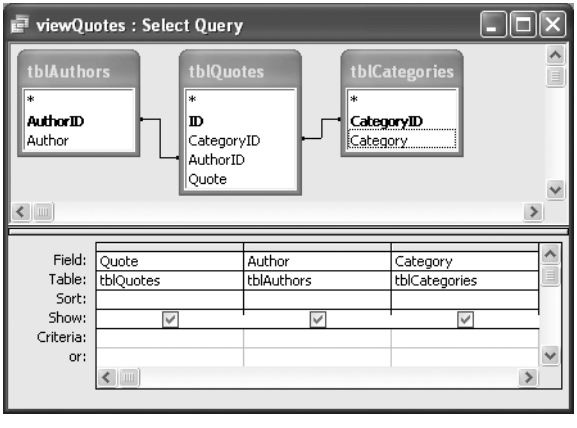


Figure 7-28. The completed query designer window

We've moved the tables around in the query designer view to make it easier to see the join lines between the three tables.

The SQL code that these actions produce uses the JOIN keyword to join the three tables together for you. It makes sure that each quote this query returns from the tblQuotes table will have the appropriate author and category retrieved from the tblAuthors and tblCategories tables.

To take a look at the SQL code that Access has produced for you, select the View menu while the query designer window is active and then select SQL View (Figure 7-29).

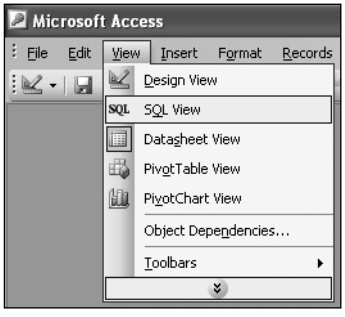


Figure 7-29. Checking out the SQL code Access has produced in SQL view

Your SQL code should look something like this:

```
SELECT tblQuotes.Quote, tblAuthors.Author, tblCategories.Category
FROM tblCategories
INNER JOIN (tblAuthors INNER JOIN
  tblQuotes ON tblAuthors.AuthorID = tblQuotes.AuthorID)
  ON tblCategories.CategoryID = tblQuotes.CategoryID;
```

Save this query in your database now by clicking the Save icon on the Access toolbar and giving it the name viewQuotes when prompted. Then close Access and return to Dreamweaver.

> *I have a habit of calling queries* Views *owing to me working primarily with SQL Server and that being the name for queries in that database. Therefore, I save all queries with a* view *prefix.*
>
> *—Rob Turnbull*

You now have everything in place to build the page that will retrieve a random quote—let's get right to it!

Displaying a random quote

In Dreamweaver, create a new ASP VBScript file and save it as randomquote.asp. The first thing you need to do on this page is retrieve the quotes from the database—all of them. Once you have them all in a recordset on the page, you'll then write some simple ASP VBScript code to pick one from the recordset at random and then display it.

Building the Quotes recordset

To build the Quotes recordset, follow these steps:

1. In the Application panel, select Bindings ➤ Recordset (Query) to begin creating the recordset.
2. When the Recordset dialog appears, enter rsQuotes in the Name field, select your database connection from the Connection drop-down menu, and then select viewQuotes from the Table drop-down menu (Figure 7-30). You want to select all the fields from the view so you can leave the rest of the dialog at the default settings (i.e., All selected in the Columns area, with Filter and Sort set to None).

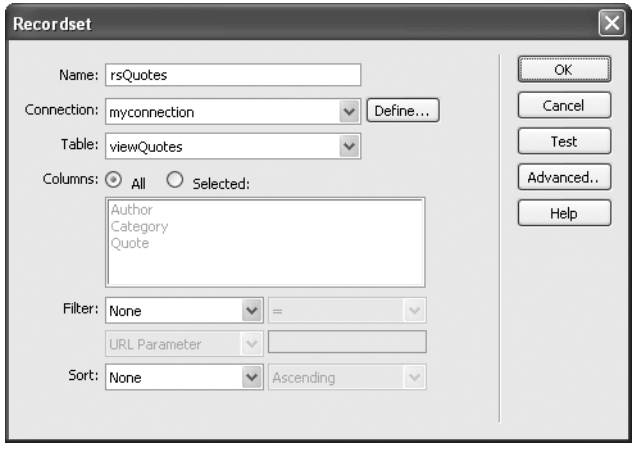


Figure 7-30. Building the Quotes recordset

3. Click OK to create the recordset and close the Recordset dialog.

An important change you need to make is to the recordset cursor type. **Cursor types** are what ADO, or ActiveX Data Objects, use to specify how the data you're retrieving from the database should be viewed by the connection retrieving it. Two of ADO's four cursor types won't work for the code you're about to create, and one of those is the default cursor type that Dreamweaver chooses for you when you create a recordset, Forward-only, so you need to change it to one that will work.

> *A good reference for ADO recordset information is the ADO section of the W3Schools website:* `www.w3schools.com/ado/ado_ref_recordset.asp`.

You can bring up the recordset properties in the Property inspector by clicking the recordset name in the Server Behaviors panel. Alternatively, you can make the same details visible in Code view.

1. Switch to Code view and then click the cursor anywhere within the recordset code near the top of the page. The Property inspector will change to display the settings of the recordset. Magic!
2. In the Property Inspector, change the Forward-only cursor type to Static. This changes the 0 to 3 in the code of the recordset.

3. Switch back to Design view and type Category:.
4. Press Enter to create a second paragraph, as this one just became the first paragraph by default.
5. Type Author: and press Enter again.
6. Finally, type Quote: into the third paragraph.

   Next, you need to bind the values you retrieve from the database to the page in the appropriate places. Dreamweaver keeps these data bindings in the Bindings panel of the Application panel group.

7. Click at the end of the Category: text and add a space.
8. In the Application panel, click Bindings, expand the rsQuotes recordset, select the Category field, and click Insert at the bottom of the Bindings panel to add that database field to the page.
9. Repeat step 8 to add the Author field to the page, and then repeat the step once more for the Quote field (Figure 7-31).

   Now you need to write the code to grab a quote at random from the recordset and display that quote's details in these three elements on the page.

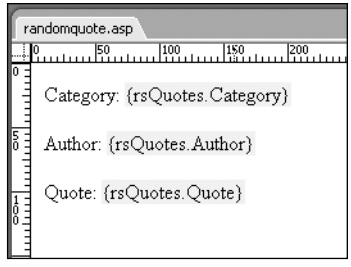


Figure 7-31. The three bindings on the page in Design view

10. Switch to Code view and locate the opening <p> tag of the first paragraph. It should be immediately below the opening <body> tag. Press Enter twice to create a space between the <body> tag and the <p> tag, and then click on the first blank line below the <body> tag.
11. The first line of code you'll add is to avoid the page throwing an error if the recordset is empty (i.e., the database doesn't have any quotes in it). Enter the following code now:

```
<%
  If Not rsQuotes.EOF And Not rsQuotes.BOF Then
```

12. The next block of code uses the number of records that have been retrieved in your rsQuotes recordset. A random number will be generated and then the recordset will be instructed to move to that record. Enter the following code now:

```
  varTotalQuotes = (rsQuotes.RecordCount-1)
  Randomize
  varRandomNumber = Int((varTotalQuotes * Rnd) + 1)
  rsQuotes.Move varRandomNumber
%>
```

This simple code is the main engine for our random quote generator, so we'll explore it in detail.

You assign the total record count minus 1 to the variable `varTotalQuotes`. You subtract 1 from the total to ensure that the code later on doesn't accidentally go past the end of the recordset, which would result in the page spitting out an error.

Next, you instruct the VBScript engine to make your random number **really** random by calling the built-in `Randomize` function.

> *Without this call to* `Randomize`*, a call to the built-in* `Rnd` *function will return the same random number quite a lot. Not very random!*

Then you assign an `Int` value (a whole number) to the variable `varRandomNumber`. This integer is generated by taking the value held in the `varTotalQuotes` variable—your total record count minus 1—and multiplying it by the number that the `Rnd` built in function gives you. The built-in `Rnd` function will always return a **real** number (decimal) greater than or equal to 0.0 and less than 1.0. You add 1 to that equation to ensure that you will always have at least 1 as a result of this sum.

Finally, you instruct the `rsQuotes` recordset to `Move` to the position in the recordset specified in the `varRandomNumber` variable. This value will never be zero, nor will it ever be one more than the total number of records available, either of which would result in an error.

That's the end of the code that you need to add to before the output paragraphs.

13. Move down to just after the end of the last of the three paragraphs you added to the page earlier, just above the closing `</body>` tag, and add this VB Script code:

```
<%
End If
%>
```

You must `End` the `If` you started earlier on or you'll get an error for that too!

The completed code for this section of the page should look like something like this:

```
<%
If Not rsQuotes.EOF And Not rsQuotes.BOF Then
  varTotalQuotes = (rsQuotes.RecordCount-1)
  Randomize
  varRandomNumber = Int((varTotalQuotes * Rnd) + 1)
  rsQuotes.Move varRandomNumber
%>
  <p>Category: <%=rsQuotes.Fields.Item("Category").Value%></p>
  <p>Author: <%=rsQuotes.Fields.Item("Author").Value%></p>
  <p>Quote: <%=rsQuotes.Fields.Item("Quote").Value%></p>
<%
End If
%>
```

Now the page is complete. When you save it and test it in your browser, you should get a result similar to that shown in Figure 7-32.

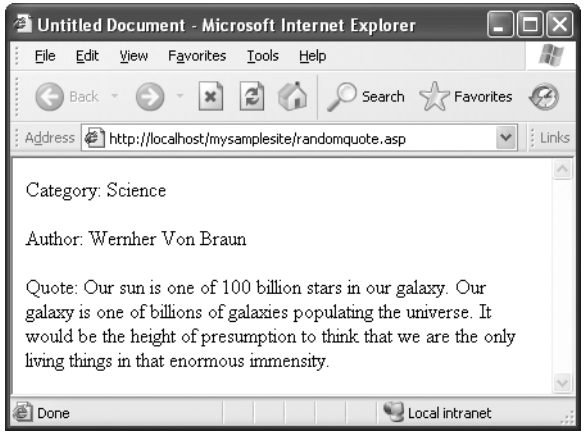

Untitled Document - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/mysamplesite/randomquote.asp
Links
Category: Science
Author: Wernher Von Braun
Quote: Our sun is one of 100 billion stars in our galaxy. Our galaxy is one of billions of galaxies populating the universe. It would be the height of presumption to think that we are the only living things in that enormous immensity.
Done
Local intranet

Figure 7-32. Displaying a random quote, along with the category and author information

Almost every time you refresh the page, a different quote will be displayed—so long as you have more than one quote in your database, of course!

> *We say **almost** every time you refresh the page in the preceding sentence because this is a **random** quote generator, and as such, it is possible for it to randomly generate the same number two (or more) times in a row.*

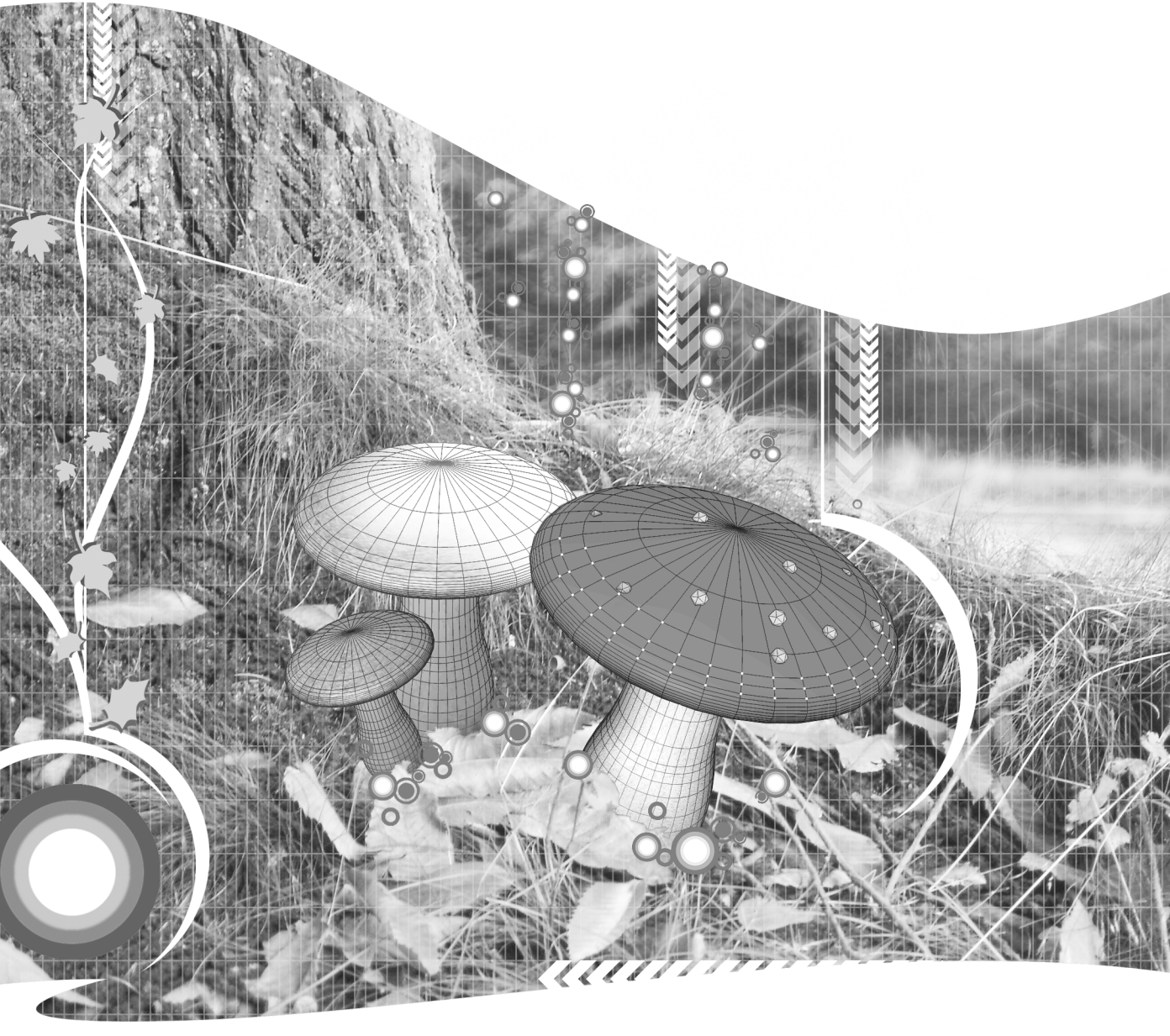
Conclusion

In this chapter you learned how to use primary and foreign keys in a database to enable relationships between the data in related tables, and you also created relationships that make use of those primary and foreign keys. In addition, you saw how to use dynamic select lists that can automatically select a value in the list based on a corresponding value from the record being edited.

This chapter also covered how to simplify moving certain server-side objects from page to page (such as copying recordsets) to speed up development time, and you examined how to easily update server behaviors to include extra elements in them when adding new fields to a form.

On top of that, you learned how to create a query in Access that brings together the data needed on the page that displays the random quote and, finally, you looked in depth at the code used to actually retrieve a random quote and display it on the web page.

In the next chapter, you'll take a look at the Dreamweaver tools that enable you to restrict access to web pages, and you'll use those tools to make the quotes administration system that you've built over the last two chapters a lot more secure.



Chapter 8

RESTRICTING ACCESS

Over the previous two chapters, you built a simple quotes administration system that uses a database to store quotes and a set of ASP pages to administer that data. All the pages you've built so far do exactly what you need them to do, but if you were to put them online for real-world use, you could end up in all sorts of trouble—at least your database could, because none of the administration pages have restricted access. This leaves them open to use by anyone who could browse to the pages.

Restricting access in Dreamweaver consists of several easy-to-use server behaviors that, once applied to your pages, will lock out any unauthorized access, leaving your data in a much safer situation! In this chapter, we'll update the quotes administration system, adding a login system to allow only authorized access to specific users. Then we'll secure all the administration pages to ensure that only those authorized users can access them.

Creating the administrators database table

Before you start to build a login system, you need to add a table to your database to store those login details in. Add a new table to the database called `tblAdmin` with the following structure.

Field name	Data type	Settings
ID	Autonumber	Primary Key
Username	Text	Field size = 50 (Default) ➤ Required = Yes
Password	Text	Field size = 50 (Default) ➤ Required = Yes
AccessLevel	Text	Field size = 50 (Default) ➤ Required = Yes

You set the `Required` property of the Username, Password, and AccessLevel fields to Yes so that these columns can't be left blank. You need to make sure that the data can't be submitted to the database unless the fields are valid (nonblank) entries. I'll cover this in more detail later in this chapter.

Save this new table, and then open it in Data Entry view to add a user (you), so that you can log in to the system built in the previous chapter. Once you've added one user, you'll then be able to log in to the system and add any number of further users from the web page. But to stop everyone from being able to register in your admin database, you'll need to protect the page that adds users to the database. This means that you'll have to be logged in to be able to add a user; therefore, you can add one user now. Enter your chosen username and password in the relevant fields, and then enter Admin in the AccessLevel field. You'll be using this AccessLevel field later, and this example uses Admin.

In SQL Server, the structure for this basic `tblAdmin` table is as follows.

Field name	Data type	Settings
ID	Int	Primary Key ➤ Identity = Yes ➤ Seed = 1 ➤ Increment = 1
Username	Text	Allow Nulls = Unchecked ➤ Size = 50
Password	Text	Allow Nulls = Unchecked ➤ Size = 50
AccessLevel	Text	Allow Nulls = Unchecked ➤ Size = 50

Having Allow Nulls unchecked in SQL Server is the same as having the `Required` property in Access set to Yes—this cannot be a blank field in the database.

With the simple `tblAdmin` table in place, you can begin to build the login system that will make use of this table.

Creating the login system

This login system is quite simple to build. It will use a form to allow a user to actually log in to the system. The form will make use of Dreamweaver's built-in server behaviors to check that the login details are correct and redirect accordingly. Once a user has successfully logged in, access will be granted to the administration pages based on the logged-in user's access level, which is stored in the database.

Building the login form

Create a new ASP VBScript page and save it in the root of your website, calling it `login.asp`. Then follow these steps:

1. Add a form to the empty page. Press Enter a couple of times to add some empty paragraphs to the page. This will save you a headache or two later in getting around a small bug in Dreamweaver.

> *The small bug just referenced inserts new paragraph tags around a form element when Enter is pressed to begin a new paragraph, separating that form element from its label.*

2. Add a text field form element in the first paragraph of the form, and in the Input Tag Accessibility Attributes dialog, enter Username in the Label field, select Attach label tag using 'for' attribute, set Position to Before form item, and enter an Access key of u and a Tab Index of 1 (Figure 8-1).

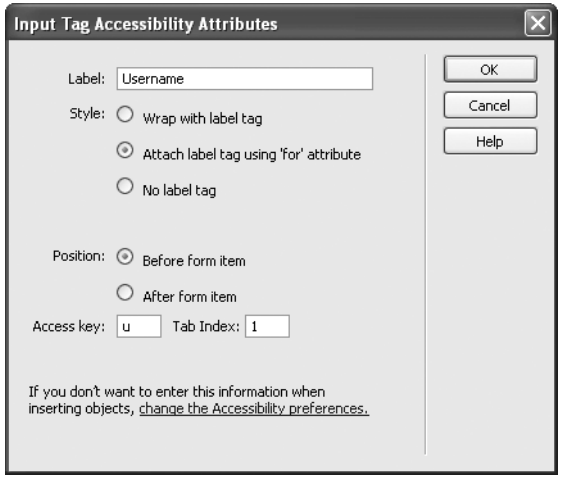


Figure 8-1. The Input Tag Accessibility Attributes dialog

3. Click OK to add the text field to the form.

4. In Design view, click below the first form element, in the second paragraph, and add a second text field form element. Use the same settings as for the Username field, except call this one Password and assign an Access key of p and a Tab Index of 2.

5. With this field selected in Design view, go to the Property inspector and select the Password radio button. This will ensure that when a user enters her password in this field, it will be masked and unreadable.

   If you didn't enter a third paragraph to start with, you can enter one now below the Password field.

6. Add a form button to the page, but leave the Label field blank. Select No label tag for this item and assign an Access key of s and a Tab Index of 3. The default values for this button will name it Submit, and it will be a Submit button, which is what we want. These attributes of the button can all be changed in the Property inspector.

   Now you'll need to dive into the code and make a few small amendments that Dreamweaver doesn't do for you.

7. Enter Code view. Copy the text of the two form elements (username and password) and paste them into the Label `for` attribute and also into the `name` and `id` attributes of both text fields.

8. You need to add a little conditional code to inform the user that a login failed when it happens. Switch to Design view and add a paragraph above the Username text field and type Login failed, please retry. You will wrap that entire paragraph in conditional code that checks for the existence of a parameter that you will add to the Log In User server behavior when you apply that to the page in the next section.

9. Switch to Code view and surround your conditional paragraph with this `if` statement:

```
<%
If Request("login ") = "failed" Then
%>
<p>Login failed, please retry</p>
<%
End If
%>
```

The code for the form on your login page so far should look like this:

```
<form id="form1" name="form1" method="post" action="">
  <%
  If Request("requsername") <> "" Then
  %>
  <p>Login failed, please retry</p>
  <%
  End If
  %>
  <p>
    <label for="Username">Username</label>
    <input type="text" name="Username" accesskey="u"
     tabindex="1" id="Username" />
  </p>
  <p>
```

```
    <label for="Password">Password</label>
    <input type="password" name="Password" accesskey="p"
      tabindex="2" id="Password" />
  </p>
  <p>
    <input type="submit" name="Submit" value="Submit"
     accesskey="s" tabindex="3" />
  </p>
</form>
```

You've just built your basic login form. Save this page and let's move on to look at how to add the Log In User server behavior.

Adding the Log In User server behavior

Currently, the form doesn't do anything. Browsing this page and clicking the Submit button simply reloads the page each time. Now you need to add the server behavior that will check the database to see if a user can log in. If the user can log in, you will log him in and redirect him to the quotes administration home page. If he cannot log in, you will redirect back to this page and display the conditional code that you added to inform the user of what is happening.

1. In the Application panel, select Server Behaviors ➤ User Authentication ➤ Log In User, and the dialog for this server behavior will open. The first three fields should be automatically filled in correctly because of the names you used for the form elements: Username and Password. If they are not, select the appropriate form elements for those fields in this dialog.
2. Select your connection in the Validate using connection drop-down list, and the other three fields in this part of the dialog will make a best guess for you. They need to be set correctly, so if they are not already, select tblAdmin in the Table list, select Username in the Username list, and select Password in the Password list.

> *The fields you've just set will be used to validate a login against the database, so you must ensure that they are mapped correctly. You will validate the* Username *field on the form against the* Username *column in the database, and so on.*

3. You can either browse to the file you want successful logins to be redirected to or you can type it in. Either way, set the If login succeeds, go to field to allquotes.asp, which is the page in the administration system that lists all the quotes in the database and will suffice as a homepage for logged-in administrators in this case. If login fails, go to should be set to login.asp?login=failed.

> *The preceding* If login fails, go to *setting tells the Log In User server behavior to redirect failed logins back to the same page, but it also passes a parameter that the conditional code on your page looks for in order to display the* Login failed, please retry *paragraph. The parameter name is one of our own choosing and is not fixed. If you want to use something different, do so, but make sure that your conditional code uses the same parameter name, or it won't work.*

Next, you want to make use of the `AccessLevel` column in the database. This will allow you to build an administration system that uses multiple access levels for its administrative users—you may want certain pages to be accessed only by a certain group of users. Access levels give you one method of doing this.

Dreamweaver will assign the access level of the user logging in to a session variable called `MM_UserAuthorization`. *This will then be checked on the pages that are restricting access, and if it doesn't exist, or a match isn't found, then access will be denied.*

4. To set up access levels in this server behavior, you need to specify that it should restrict access based on Username, password, and access level and that it should get the level from the `AccessLevel` column in the database. Your completed dialog should look like Figure 8-2.

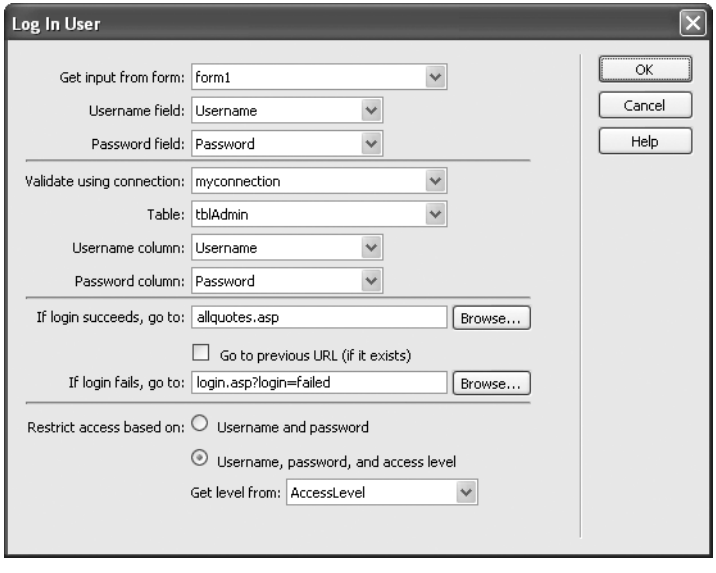


Figure 8-2. The completed Log In User dialog

5. Click OK to apply the server behavior to the page. Close the dialog, and then save your page.

This server behavior creates up to two session variables for you to make use of in your site: `MM_Username` *and* `MM_UserAuthorization`. `MM_UserAuthorization` *is created only if you opt to use the access levels feature this server behavior offers. If you don't use that feature, then it creates only the* `MM_Username` *session variable. In the "Expanding the Log In User server behavior code" section later in this chapter, you will explore how easy it is to add your own session variables to this setup, on top of those already created for you.*

If you browse this page now, and you've added yourself to the database as the first user, you can now log in to the system—although at this stage that's all you can do!

You can also test the failed login conditional code by entering a username and password combination that doesn't exist in the database (Figure 8-3).

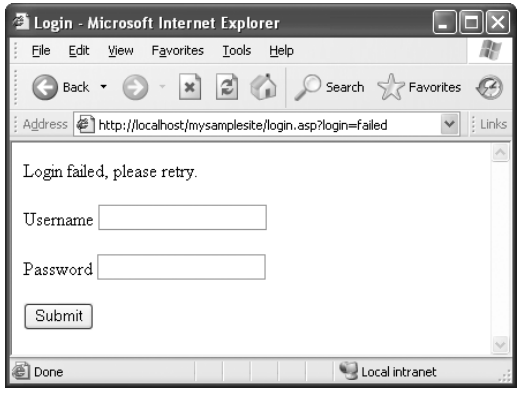


Figure 8-3. Testing the conditional login failed code in the browser

With the login form completed, you can now add some more code to the rest of the administration pages to restrict access to them.

Restricting access

The page you've coded your login page to redirect to on successful login is the `allquotes.asp` page, so it makes sense for you to start restricting access there so you can see how this works in practice.

1. Open the `allquotes.asp` page. In the Application panel, select Server Behaviors ➤ User Authentication ➤ Restrict Access To Page.
2. Restrict based on should be set to Username, password, and access level. Click the Define button to define a list of acceptable access levels. When you click Define, the Define Access Levels dialog opens.
3. You're adding only a single access level at this point, so just type Admin in the Name field and click OK to close this dialog (Figure 8-4).

Figure 8-4. Defining an access level

If you need to add further access levels to this dialog, click the + button above the list of defined levels and then enter the name in the Name *field. You can repeat this process as often as is necessary for your application. For the purposes of this example, you just need one level for now.*

4. With the access level defined and now appearing (selected) in the Select level(s) list in the Restrict Access To Page dialog, all that's left to do is define what should happen if an unauthorized attempt to access this page occurs. In the If access denied, go to field, type login.asp or click the Browse button and select login.asp. Your completed dialog should look like Figure 8-5.

Figure 8-5. The completed Restrict Access To Page dialog

5. Click OK to apply this server behavior to the page and close the dialog.

Now if you try to browse the `allquotes.asp` page directly, without being logged in first, you will be redirected to the `login.asp` page (Figure 8-6).

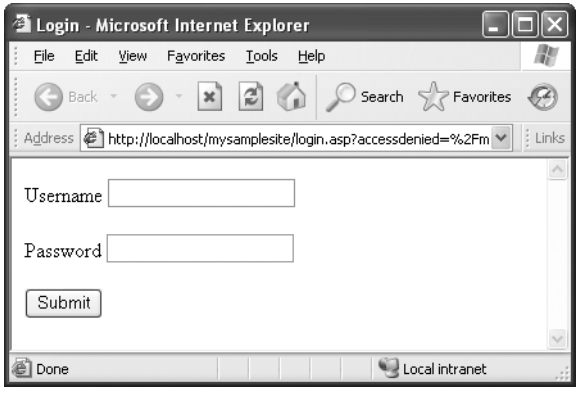


Figure 8-6. Arriving at the login form from a protected page

You might also notice that in the address bar of the browser, the Restrict Access To Page server behavior has added a parameter to the URL called accessdenied with a value of the address of the page that has sent you to the login page. It is URL encoded to help prevent problems that can arise from having slashes in the URL where they shouldn't be, but you can see by reading it that the page name allquotes.asp is in there.

> *You could quite easily make use of this piece of information to generate a more user-friendly login system that takes you back to the page you were trying to access before being logged in. In fact, you'll look at exactly that in the upcoming "Creating a user-friendly login" section.*

The next step in securing this admin system is to apply the Restrict Access To Page server behavior to all the other admin pages. In the admin system you built over the previous two chapters, these pages include `insert.asp`, `insertauthor.asp`, `insertcategory.asp`, `editquote.asp`, and `delete.asp`.

You will notice that when you go to add this server behavior to the next page, the Select level(s) list already contains the Admin entry you entered in the dialog the first time you used it. You will only need to click that entry in the list to select it when applying this server behavior to the other pages.

> *Dreamweaver keeps a note of the access levels that you have entered into that list in the background to save you from having to remember them or type them in multiple times. If you add more access levels to the list at any point, that list will always contain the full list of access levels regardless of which page you add the server behavior to—so you could easily go back to a page that already has this server behavior applied to it and update it to use a new access level that was thought about later, simply by clicking the new level in the* Select level(s) *list.*

The next server behavior you'll make use of is called Log Out User.

Adding the Log Out User server behavior

This is a fairly simple server behavior that just removes the session variables created when you log in. Let's go ahead and apply it to the `allquotes.asp` page—as this page is the main hub of the administration system, it's a logical place to add the behavior. You could add it to all your pages, of course, but in this instance, there's really no need.

1. Open the `allquotes.asp` page. Add a new paragraph to the top of the page, and then in the Application panel select Server Behaviors ➤ User Authentication ➤ Log Out User.
2. Leave the Log out when option set to Link clicked. The default setting it offers to create a new link on the page for you is fine. It will create a piece of text, Log out, in the paragraph you just created, so long as the cursor was inside that paragraph when you began to apply this server behavior (it will place the link wherever your cursor was on the page).

3. You will force a redirect to the `login.asp` page when the Log out link is clicked, so type login.asp in the When done, go to field, or click the Browse button and browse to that page. With your page coded the way it is, you could simply leave this field blank and the redirection that occurs when you try to access this page when you're not logged in would take effect, but you'll force the redirect anyway, just to speed up the code execution. Your completed dialog should look like Figure 8-7.

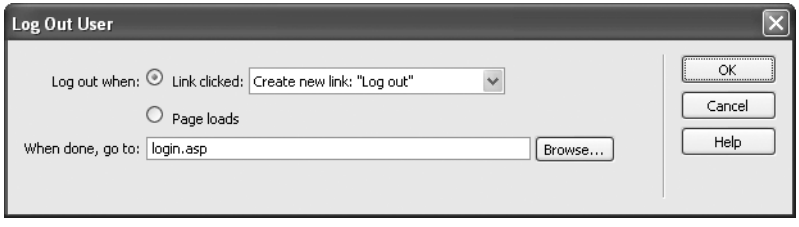


Figure 8-7. Creating the Log Out User link

4. Click OK to apply the server behavior to the page and close the dialog. The server behavior will create the Log out link text on the page for you.
5. While you're adding links to the `allquotes.asp` page, also add a link to the `adduser.asp` page, which you'll create next. Click after the Log out link on the page and then type Add user. Select the Add user text in the Property inspector and type adduser.asp in the Link field.
6. Now save and browse the page (don't forget to log in first!).

The `allquotes.asp` page with the Log out and Add user links should look like Figure 8-8.

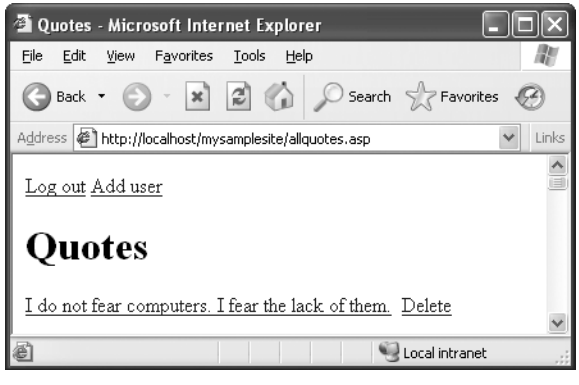


Figure 8-8. The two new links on the page

Clicking the Log out link removes the session variables that the Log In User server behavior creates when you log in, and then redirects you back to the `login.asp` page, as you instructed it to do.

Now that you have the basic restricted access system set up, you can create a new secure page that will allow you to add new admin users to the database.

Registering new users

In this section, you'll build a simple form to collect a new user's details, which will consist only of Username and Password. You'll also assign the new user to an administrative access level, which will be admin. This will be used to specify the user's access level in the secure system and is case sensitive, meaning that Admin (with an initial capital A), for example, would not be classed as the same access level and validation would fail.

First, create a new ASP VBScript page called `adduser.asp` and save it to the root of your site. Then follow these steps:

1. Add a heading to the page with the text Add user. Add a form to the page and then insert three text fields: Username, Password, and Access Level. We recommend you get into the habit of using the accessibility features when they are presented to you—it's very good practice!
2. With your three text fields appropriately labeled, go into Code view and change the default names Dreamweaver assigned to them (textfield, textfield2, and textfield3) to something more appropriate. Change the name and ID of each of the fields to Username, Password, and AccessLevel, respectively, and ensure that you do not use spaces in the names.
3. Click to select the Access Level text field in the Property inspector and type Admin into the Init val field. This will give the form element the default value of Admin, which will save you from typing it in each time you add a new Admin user.
4. Add a button called Submit to the form below the three text fields. (Note that Dreamweaver labels this one correctly for you!) Your code should look like this so far:

```
<form id="form1" name="form1" method="POST"
     action="<%=MM_editAction%>">
   <h1>Add user</h1>
   <p>
     <label for="Username">Username</label>
     <input type="text" name="Username" accesskey="u"
       tabindex="1" id="Username" />
   </p>
   <p>
     <label for="Password">Password</label>
     <input type="text" name="Password" accesskey="p"
       tabindex="2" id="Password" />
   </p>
   <p>
     <label for="AccessLevel">Access Level</label>
     <input name="AccessLevel" type="text" id="AccessLevel"
       accesskey="a" tabindex="3" value="Admin" />
   </p>
   <p>
     <label for="Submit">Submit</label>
     <input type="submit" name="Submit" value="Submit"
       accesskey="s" tabindex="4" id="Submit" />
   </p>
   <input type="hidden" name="MM_insert" value="form1">
</form>
```

The form in Design view should look like Figure 8-9.

5. The next thing you need to do is add the Insert Record server behavior to the page. In the Application panel, select Server Behaviors ➤ Insert Record.

6. In the Insert Record dialog, select your connection from the Connection drop-down list. Choose tblAdmin from the Insert into table drop-down list. Leave the After insert, go to field blank, as you can simply come back to this page after inserting, and leaving this field blank will do exactly that. Get values from form1 is correct by default (unless you sneakily added another form to the page while we weren't looking!). Finally, the three form elements should all be set to insert their data into the correct column using the correct data type by default.

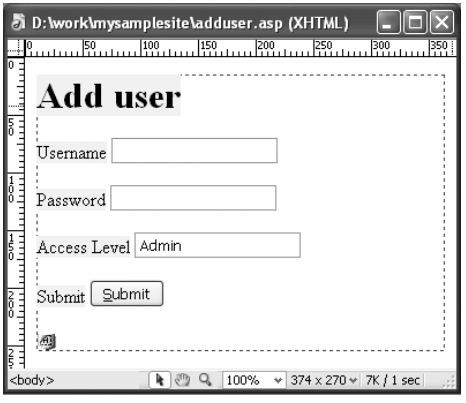


Figure 8-9. The Add user form in Design view

Our clever use of form element names that match up with the database column names saved a little work here because Dreamweaver always tries to take a best guess!

Your completed Insert Record server behavior dialog should look like Figure 8-10.

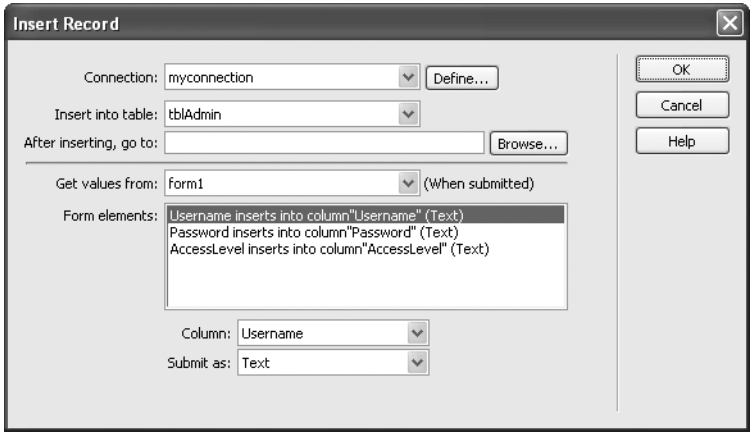


Figure 8-10. The completed Insert Record dialog to add a new administrative user

Because you've chosen to return to this page after inserting a record, you need to add a link back to the quotes list page. Add a new paragraph below the Submit button and type Back to the quotes list. Then link that text to the `allquotes.asp` page by selecting it all, clicking the target icon in the Property inspector, dragging it to the `allquotes.asp` page in the Files panel, and then releasing it. Alternatively, just type allquotes.asp in the link field.

Checking the new username

To keep your system running smoothly, it is important to ensure that each user who can log in has a unique username and password combination. To accomplish this, you simply need to enforce that one of those two items is unique in the database for every user, which you can do by making use of the Check New Username server behavior.

> *You don't have to actually specify that the username be the unique item if you don't want to—you could just as easily choose to make everyone have a unique password if you like, but Macromedia had to call this server behavior* ***something****!*

1. To begin applying this server behavior to the page, in the Application panel select Server Behaviors ➤ User Authentication ➤ Check New Username.
2. Select Username in the Username field drop-down list, and type adduser.asp into the If already exists, go to field or click the Browse button and browse to that file (Figure 8-11).

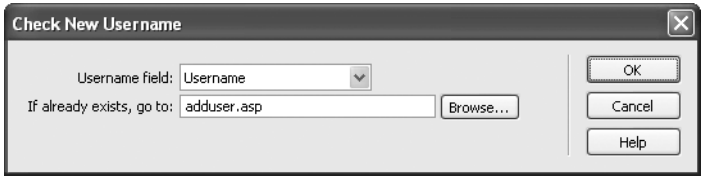


Figure 8-11. The completed Check New Username dialog

> *The field you choose in the* Username field *drop-down will be the form element on the page that will be used to check against the related column in the database for uniqueness. So, if you select* Username*, this server behavior will check the value you enter in the* Username *field on the form against all entries in the* `Username` *column of the* `tblAdmin` *table to make sure it doesn't already exist.*

If the username (or other unique field that you require and specify here) entered already exists in the database, the page will redirect back to itself, as you have instructed it to do, but it will also pass along a parameter for you to make use of on your page if you so choose. This parameter is called `requsername`, and it contains the duplicate value that is already in use in the database. You can use this parameter to add some conditional code to the page to keep you informed of what is happening.

3. Insert a new paragraph above the Username text field, below the Add user heading, and type Username () already in use, choose another.
4. Switch to Code view and add the conditional code around that paragraph as follows:

```
<% If Request("requsername") <> "" Then %>
  <p>Username (<%=Request("requsername")%>) already in use,
        choose another</p>
<% End if %>
```

Notice that you also add some code inside the brackets in the conditional text. That piece of code will display the username that was rejected for the new user registration due to it already being in the database.

Finally, you need to add the Restrict Access To Page server behavior to this page. See the "Restricting access" section earlier in this chapter for details of how to do that if you're not sure.

When you browse to this page (click the Add user link on the `allquotes.asp` page), ensuring you're logged in first, you'll see the form ready to be completed (Figure 8-12). If you try to add a new user that has the same username as one already in the database, the conditional code should display your conditional paragraph, which will include the username that you entered.

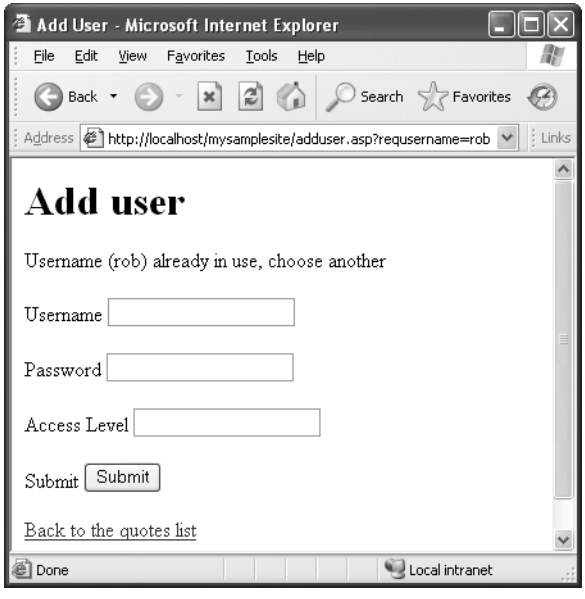


Figure 8-12. Testing the check new username code in the browser

If you enter unique details and submit the form, the new user will be added to the database.

Creating a user-friendly login

You've already seen how to simply prevent access to a page and force users to log in by redirecting them to the login form. But wouldn't it be much nicer for the user if, after having been forced to log in, she were taken straight back to the page she initially tried to access?

You can add a little bit of code to your login page to make this happen, thanks to the Restrict Access To Page server behavior providing you with a useful bit of information. The parameter that is sent with the user to the login page when she tries to access a secure page without first being logged in is called `accessdenied`, and it contains the value of the page that sent the user to the login page. All you need to do is use that parameter in your form to make the user experience a whole lot better!

1. Open the login.asp page and add a hidden form element to the form next to the Submit button.
2. In the Property inspector, type redirect in the Hidden Field field for this hidden form element (which is the name of the field), and enter a value of <%=Request("accessdenied")%> in the Value field. This will automatically populate this hidden field with the value, if any, passed to the page in the accessdenied parameter.
3. Next, you need to tweak the Log In User server behavior code, so switch to Code view and locate this line of code (on or around line 10):

   ```
   MM_redirectLoginSuccess="allquotes.asp"
   ```

   You need to add some conditional code around this line to make use of your new hidden form elements value, if it exists. If it does not exist, the normal redirect should take place.
4. Replace that current single line of code with the following conditional statement:

   ```
   If Request("redirect") <> "" Then
     MM_redirectLoginSuccess=Request("redirect")
   Else
     MM_redirectLoginSuccess="allquotes.asp"
   End If
   ```

As you can see, the variable that the Log In User server behavior uses to redirect upon successful login still exists in the new code, but there are two options to it now. If the redirect hidden form element contains a value when this form is submitted, that value will be used and the user will be taken back to the page she initially tried to access. Now that's much more user-friendly!

Expanding the Log In User server behavior code

There are times when you might need to carry more information for each logged-in user on your website than just his username and access level, which are the only two session variables that the Log In User server behavior creates for you by default.

For example, carrying the user's ID number in a session variable can be very useful if your application uses the ID number to identify who a record belongs to in the database. Fortunately, adding another session variable is quite straightforward.

1. Open the login.asp page and, in Code view, locate the line of code that defines the source for the MM_rsUser recordset:

   ```
   MM_rsUser.Source = "SELECT Username, Password"
   ```

2. You'll need to add the additional fields you want to retrieve from the database to the front of that SELECT statement (i.e., before the Username column). Let's go with the example of storing the ID number of the user in a new session variable, so type ID (which is the ID number column in the tblAdmin table) before Username in that SELECT statement. It should look like this when you've finished:

   ```
   MM_rsUser.Source = "SELECT ID, Username, Password"
   ```

3. Next, locate the line that creates the username session variable, about nine lines of code further down:

```
Session("MM_Username") = MM_valUsername
```

4. Add a new blank line the current username session variable line of code and type your new session variable creation code, as follows:

```
Session("svUID") = MM_rsUser("ID")
```

This new session variable will be called `svUID`, and it will take its value from the ID column you have retrieved from the database in the `MM_rsUser` recordset.

> *I always prefix my session variables with* `sv`*, just so I know what I'm referencing in those larger, more complex developments.*
>
> *—Rob Turnbull*

Notice that each session variable is created in the order in which it is retrieved in the recordset—that is, `svUID` is created first using the ID column of the recordset, followed by `MM_Username`, and then `MM_UserAuthorization`. Assigning the fields of the recordset must occur in the order in which they are retrieved into the recordset or an error will occur.

> *The recordset fields must be assigned in the order in which they are retrieved because the forward-only cursor type is being used on the recordset. You could change the cursor type in the code (to* `3` *for a static cursor) so this issue doesn't arise, but it's better practice to simply collect and assign everything in its correct order.*

With those code changes taken care of, save and close your page. Now when you log in, you'll create the extra session variable called `svUID`. You can easily check that this is happening by writing that session variable's value out to the page somewhere, such as on the `allquotes.asp` page.

Select the location you want to output it to and add the following code to display it:

```
<%=Session("svUID")%>
```

As you can see, we prefixed ours with the text `UserID:` to identify it clearly at the top of the page (Figure 8-13).

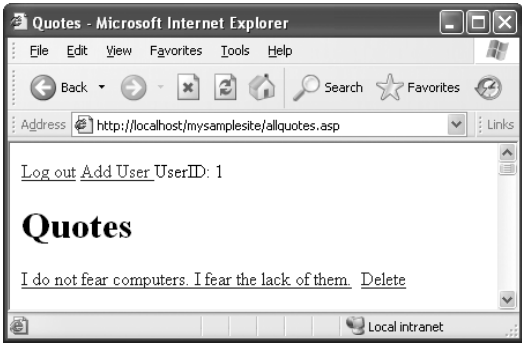

Quotes - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/mysamplesite/allquotes.asp
Links
Log out Add User UserID: 1
Quotes
I do not fear computers. I fear the lack of them. Delete
Local intranet

Figure 8-13. Displaying the UserID session variable value on the page

Implementing the "remember me" feature

Something you'll see on login forms is a **remember me** feature. This feature is becoming a rarity, though, as web browsers tend to offer to remember website login details for you these days, but it's still a very useful feature to incorporate into your form.

A remember me feature makes use of cookies that will be stored on the user's computer; therefore, it will only work on those users' computers that allow you to store cookies on them. If the user has disallowed cookies, this feature will not work for him.

Updating the login form

To update the login form, follow these steps:

1. Open the `login.asp` page and add a check box between the Password field and the Submit button.
2. Enter a name of Remember in the Label field of the Input Tag Accessibility Attributes dialog. Select Before form item as the Position and give it an Access key of r and a Tab Index of 3. Your completed dialog will look like Figure 8-14.

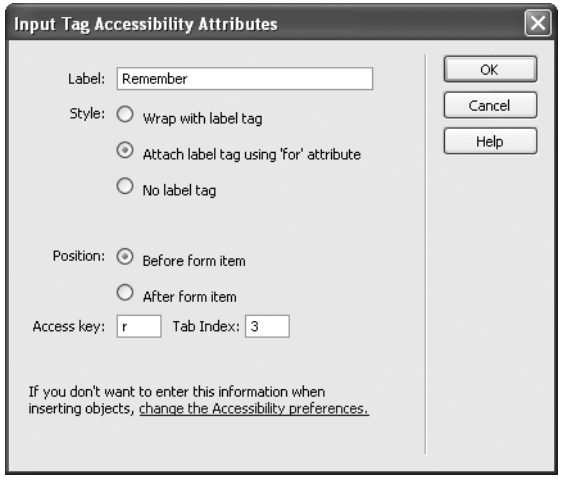


Figure 8-14. Adding a remember me check box to the form

3. In the Property inspector, give the check box a Checked value of Yes. You will be using this value later when you come to check the stored cookie value of this check box.
4. Switch to Code view and rename the `name` and `id` attributes of the check box to `Remember`, and rename the `for` attribute of the `Label` tag to `Remember` also.

> *You will need to change your* Submit *button* Tab Index *to* 4 *(from* 3*) so it does not conflict with the* Remember *check box tab order.*

5. Next, you'll add some cookies to the Bindings panel. In the Application panel, select Bindings ➤ Request Variable. In the Request Variable dialog that appears, select Request.Cookie from the Type drop-down list and enter ckUsername in the Name field (Figure 8-15).

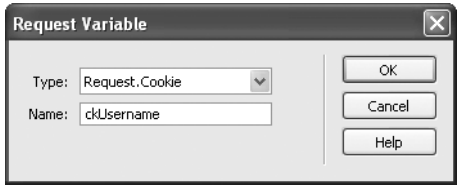


Figure 8-15. Creating a username cookie in the Request Variable dialog

6. Click OK to create the username cookie request variable.
7. Repeat this process to create two more cookies: `ckPassword` and `ckRemember`.

   With all three request variables created, your Bindings panel should look like Figure 8-16.
8. Switch back to Design view and select the Username text field.
9. In the Property inspector, click the lightning bolt icon next to the Init val field to launch the Dynamic Data dialog. Your newly created request variables will be listed there. Select ckUsername and click OK to close the dialog (Figure 8-17).

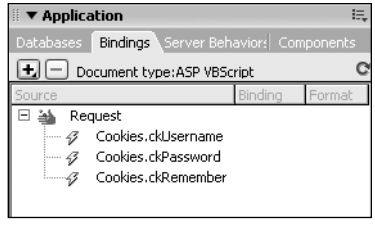


Figure 8-16. Three request variables in the Bindings panel

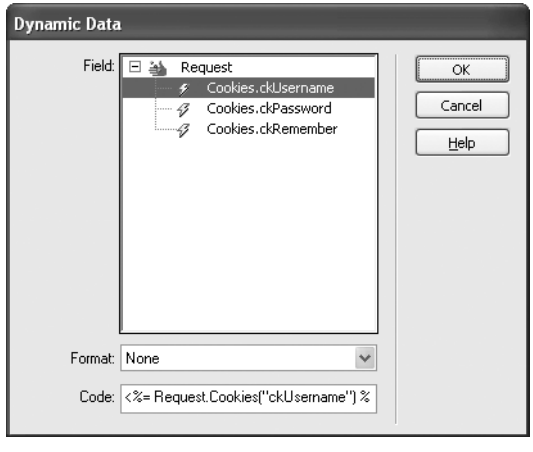


Figure 8-17. Binding the username cookie to the username form element so the form can remember it

The Init val field in the Property inspector will be populated with the correct ASP code, `<%= Request.Cookies("ckUsername") %>`, and the Username text field in Design view will also show a dynamic data binding.

If your preferences are set to show more than the curly braces only, then you should see `{Cookies.ckUsername}` *in the* Username *field.*

10. Perform the same binding to the Password field and, once you've finished, click to start the binding process on the Remember check box.

    The check box does not allow data binding the way a text field does. Instead, you make it a dynamic check box, and you will be checking the value stored in the `ckRemember` cookie to determine whether this dynamic check box should be checked or not.

11. With the check box selected in Design view, click the Dynamic button in the Property inspector to open the Dynamic CheckBox dialog. Make sure the Remember check box is selected in the CheckBox field, and then click the lightning bolt icon next to the Check if field to launch the Dynamic Data dialog and select ckRemember from the Request variables list.

12. Click OK to close the Dynamic Data dialog and then, back in the Dynamic CheckBox dialog, type yes in the Equal to field. Your completed Dynamic CheckBox dialog should look like Figure 8-18.

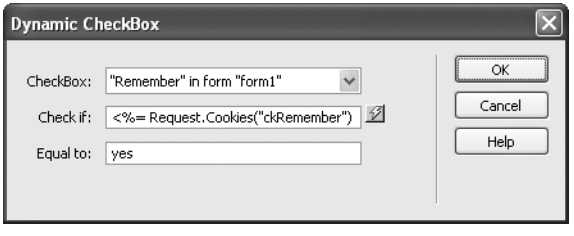


Figure 8-18. Making the remember me check box remember its own settings based on the stored cookie

13. Click OK to apply the server behavior to the page.

Finally, you need to write the code to actually create the cookies when the form is submitted.

Baking the cookies

Switch to Code view and add the following code to the top of the page, starting on line 3, immediately after the page language declaration and the connection include:

```
<%
If Request("Submit") <> "" Then
  If Request.Form("Remember") = "yes" Then
    Response.Cookies("ckUsername") = Request.Form("Username")
    Response.Cookies("ckUsername").Expires = Date + 30
    Response.Cookies("ckPassword") = Request.Form("Password")
    Response.Cookies("ckPassword").Expires = Date + 30
    Response.Cookies("ckRemember") = Request.Form("Remember")
    Response.Cookies("ckRemember").Expires = Date + 30
  Else
```

```
        Response.Cookies("ckUsername") = ""
        Response.Cookies("ckUsername").Expires = Date - 1
        Response.Cookies("ckPassword") = ""
        Response.Cookies("ckPassword").Expires = Date - 1
        Response.Cookies("ckRemember") = ""
        Response.Cookies("ckRemember").Expires = Date - 1
      End If
    End If
    %>
```

This code first checks if the form has been submitted by checking that the value of the `Submit` request item is greater than "nothing." If this is true, then a check is made to see if the Remember check box is selected. If it is, the cookies are created and given a shelf life of 30 days from the date they are created (today). If the check box is not checked, then the cookies are set to nothing and are set to expire yesterday, which basically removes them.

> *If you don't want to go to all the trouble of typing in this code, download a* ***free extension*** *from Rob's website,* `www.robgt.com`*, that writes this code for you. The extension is called* ***LoginCookies*** *and it's located in the* Products *section of the website (and yes, this extension works with Dreamweaver 8).*

You can now save and browse the web page. When you log in and select the Remember check box, the next time you browse to the login form, your login details will be automatically entered for you (Figure 8-19)!

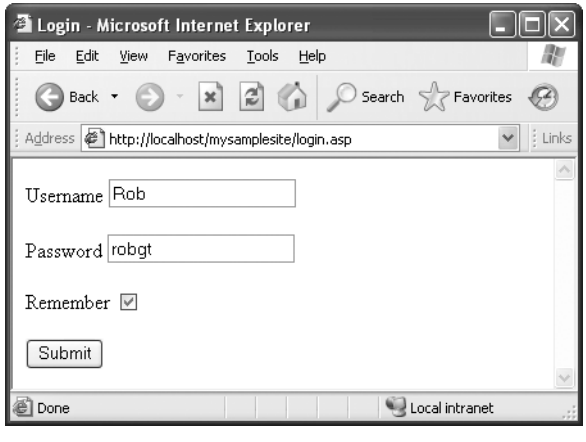


Figure 8-19. The login page remembering login details

However, if you deselect the Remember check box and log in, your details will not be remembered for the next time you browse to the login form.

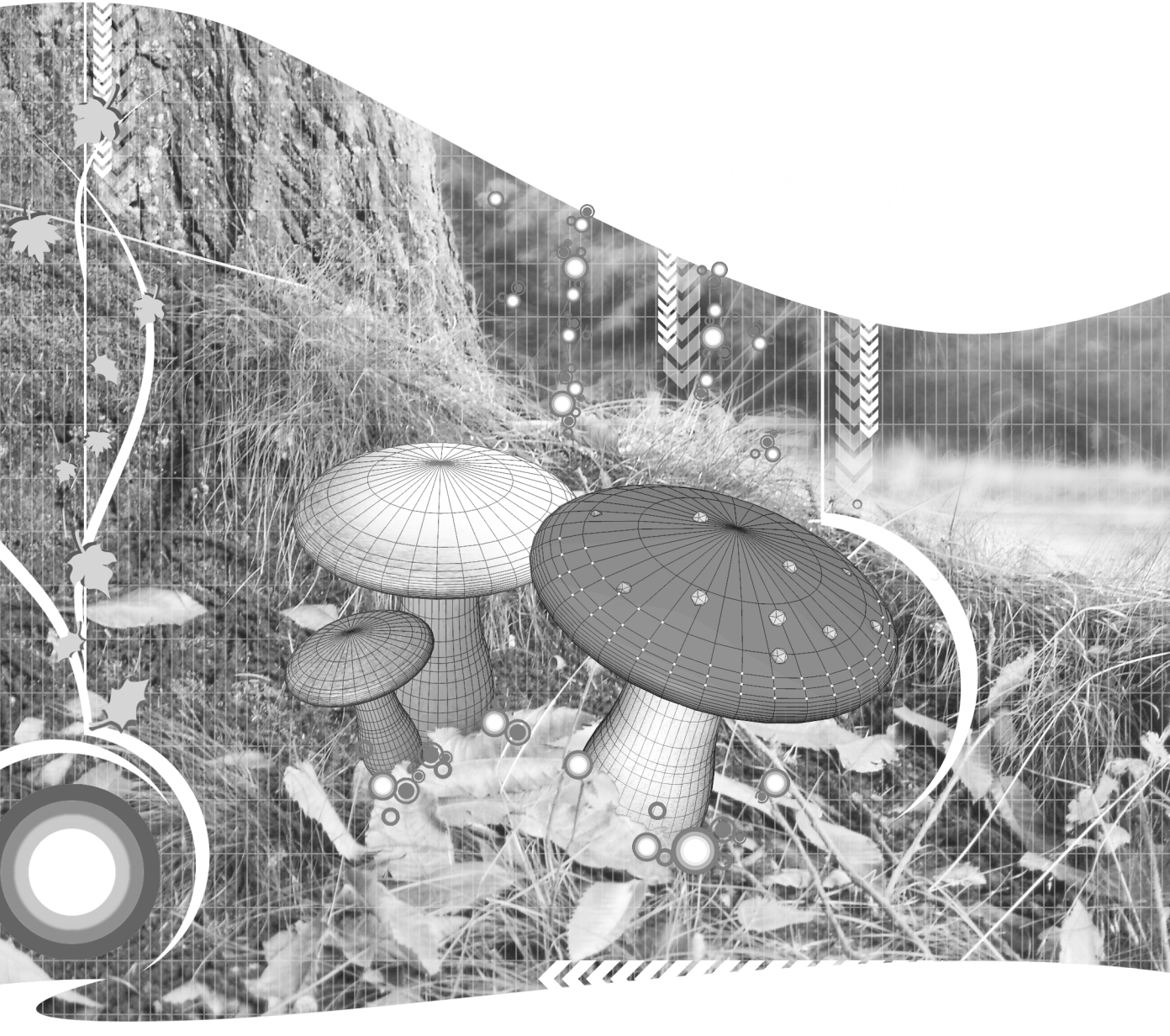
Conclusion

In this chapter, you created a login system using the Log In User server behavior, and you used the Restrict Access To Page server behavior to prevent unauthorized access to all of the quotes administration system pages. You also made use of the Log Out User server behavior to provide an easy way to remove the session variables created when you log in.

You built a new user creation page that makes use of the Check New Username server behavior, ensuring that no two users in the database will have the same username and password combination, and you added code to the login form that offers to "remember" users' login details for them using cookies stored on their computer.

Finally, you expanded upon the code that Dreamweaver writes for you in the Log In User server behavior to redirect users back to the page they tried to access without logging in first (a very user-friendly addition!), and you also added an extra session variable to those created for you by Dreamweaver's code, expanding upon the level of complexity you can add to your own developments in future.

In the next chapter, you'll look at creating a searchable website that uses some neat coding techniques to help bring search results to life and move you beyond the basics that the built-in server behaviors provide.



Chapter 9

CREATING A BLOG

You must have seen many blogs on your travels around the Internet—everyone with an ounce of computing knowledge seems to have one these days, and blog-reading is now regarded as an important way of finding out all the latest news about whatever it is you're interested in.

According to `www.dictionary.com`, a "blog" is "an online diary" or "a personal chronological log of thoughts published on a Web page." Blogs allow you to maintain a log of thoughts, ideas, or comments, they're usually ordered by date, and they're often set up so others can post comments on the original postings. Originally "Web log" or "weblog," the word has since been shortened to "blog."

Companies now use blogs so that owners and managers can communicate company news to employees, and employees can communicate with each other. Many company websites have their own private blog section, updated daily, that employees can log in to to receive informational updates about the company.

By now, you're probably thinking, "Wow—I'd love to have my own blog!" Well, you have a choice—you can either use a prebuilt, third-party solution, such as Textpattern (`www.textpattern.com`); or archive your thoughts on a dedicated blog site created by someone else, such as WordPress (`http://wordpress.com`) or Blogger (`www.blogger.com`); or, for a touch of extra class and a whole lot more freedom, you can build your own. And this chapter shows you just how to do that, using ASP and SQL Server.

Blogging application overview

From a technology point of view, a blog is merely a set of pages that allows you to insert text into a database. That text is then displayed, usually in descending chronological order.

The blog you'll be building in this chapter will allow you to maintain a chronological log of news for users to read. You'll be the only one who can post blog entries. This blog will be a way for you as a webmaster to communicate with your web users.

This application will be called (O) News—it will consist of a total of eight pages. The application will be divided into five major components, each of which you'll create individually. The five components are as follows:

- The database
- The blog display
- Administrative controls
- Administrative login/logout
- The search function

In the sections that follow, you'll take a look at each component in detail.

The database

The database is one of the most important parts of your blog—it is where all the data that makes up the blog postings is stored. The database will consist of two tables: one to store the blogs and another to store the users. You'll create the database in SQL Server. See Chapter 4 for more details about how to install SQL Server.

> *Note that you can use Access if you wish, instead of SQL Server, but there will be some minor data type differences to observe when creating your blog database. Refer to Chapter 4 for details of these differences.*

The blog display

This section will consist of three pages. The main page will display the titles of today's blogs. The user will be able to click a specific title and be taken to a details page, which will display the specific blog entry's content, title, date, and body. The third page will be an archives page, in which the titles of all blogs will be displayed. Again, the user will be able to click a specific title to display all the information for that blog entry.

Administrative controls

This section will consist of two pages that will allow you to insert, update, or delete a blog. These two pages will be restricted to administrators only.

Administrative login/logout

This section will consist of two pages. The first page will allow you to log in for access to the administrative controls; the second page will allow you to log out, destroying the session created by Dreamweaver from the login page so that the administrative pages are no longer accessible. The login details of each administrator will be stored in one of the database tables.

The search function

The search function consists of a modification to the blog's display pages; it will include a search form and a new page that will show the results of the search. When a search string is typed into the form and submitted, the results will be displayed on the results page.

> *For testing purposes, this blog application is available at my website,* `www.elbaga.net`. *Visit the site for details.*

Creating your blog database

Now you'll begin creating your blog application, so pay close attention. You'll start by creating the database first.

1. Fire up SQL Server Service Manager to make sure SQL Server is actually running. Go to Start ➤ All Programs ➤ Microsoft SQL Server ➤ Service Manager and make sure Start/Continue is selected, as shown in Figure 9-1.

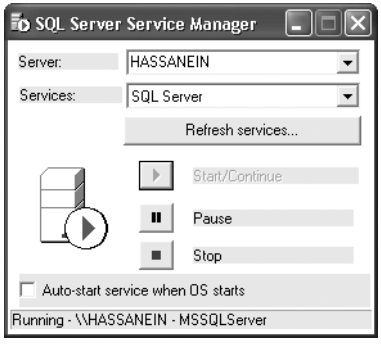


Figure 9-1. View of SQL Server Service Manager, which you can use to start and stop the server

2. Next, open SQL Server Enterprise Manager and create the database. Select Start ➤ All programs ➤ Microsoft SQL Server ➤ Enterprise Manager. From the console root, expand the menu until you see the Databases folder (the path to the Databases folder is Microsoft SQL Servers ➤ SQL Server Group ➤ (LOCAL) Windows NT ➤ Databases). Right-click the Databases folder and select New Database from the menu (see Figures 9-2, 9-3, and 9-4).

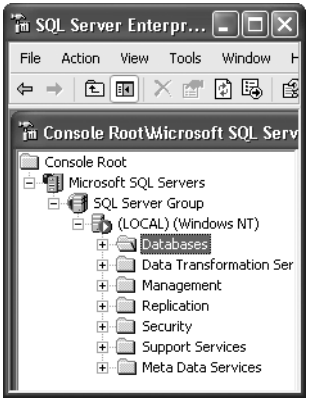


Figure 9-2. View of the console root in SQL Server

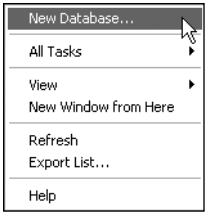


Figure 9-3. View of the shortcut menu that appears when you right-click your database in SQL Server

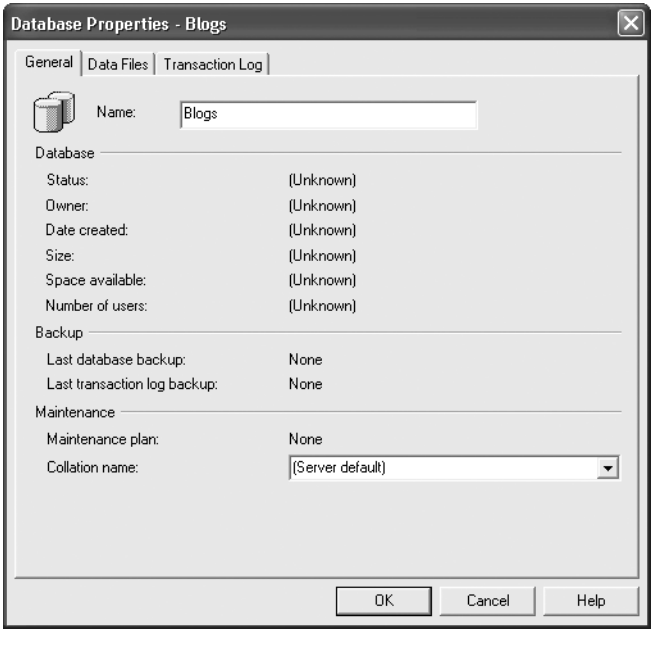


Figure 9-4. View of the Database Properties window in SQL Server

3. In the Database Properties dialog box, type Blogs (the name of your database) in the Name field. Click OK. Your database will now appear in the Databases folder.

Creating the database tables

The next step is to create your two database tables—one labeled `tbl_onews`, which will store the blogs, and the other labeled `tbl_users`, which will store the users.

Table 1: tbl_onews

1. Right-click the name of your database from the console root and select New ➤ Table.
2. Create the columns as listed in the following table:

Column name	Data Type	Length	Allow Nulls
onews_id	int	4	
onews_title	varchar	100	
onews_body	varchar	8000	
onews_dtstamp	datetime	8	
onews_online	bit	1	

3. Set the `onews_id` column to be the primary key by right-clicking the column and selecting Set Primary Key.
4. Set the Identity value for the `onews_id` column to Yes—this will ensure that the database gives each record a unique ID number.
5. Set the default value for the `onews_dtstamp` column to `(getdate())`, and the default value for the `onews_online` column to `1`. This will make sure that, whenever a new record is added, the date and time are automatically created from within the database, which will be based on the server's time. Your application will only display blogs with a value of `1` in the `onews_online` column, giving you an easy way to take a blog offline without having to delete it (by changing this value to `0`). You'll learn more about this when you create your actual pages in Dreamweaver. This default value will ensure that the blog is automatically live on your website.

Your table should now look like Figures 9-5 and 9-6. Save the table as `tbl_onews`.

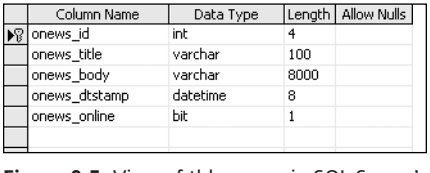


Figure 9-5. View of tbl_onews in SQL Server's Design view after the table has been created

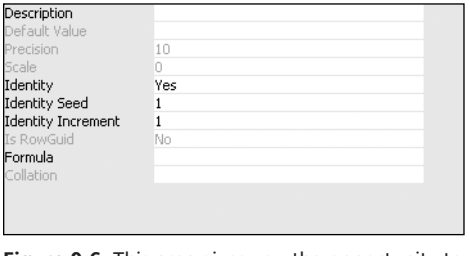


Figure 9-6. This area gives you the opportunity to specify more details about the database column.

Table 2: tbl_users

1. Right-click the name of your database from the console root, and select New ➤ Table.
2. Create the following columns:

Column name	Data Type	Length	Allow Nulls
user_id	int	4	
username	char	10	
user_pwd	char	8	
dtstamp	datetime	8	
user_level	char	5	

3. Set the user_id column to the primary key by right-clicking the column and selecting Set Primary Key.
4. Set the Identity value for the user_id column to Yes; this will ensure that the database gives each record a unique ID number.
5. Set the Default Value for the dtstamp column to (getdate()), and set the default value for the user_level column to basic. Setting this value to basic ensures that every user added to this database will receive basic membership unless admin is specified manually by the database administrator.

Your tbl_users table should now look like Figure 9-7; save the table as tbl_users.

	Column Name	Data Type	Length	Allow Nulls
▶	user_id	int	4	
	username	char	10	
	user_pwd	char	8	
	dtstamp	datetime	8	
	user_level	char	5	

Figure 9-7. View of tbl_users in SQL Server's Design view after customization

Populating the database tables

You should populate the database with some default data to ensure you have some data to display. I'll walk you through this now.

Populating tbl_onews

From the Tables section, right-click the name of your table, `tbl_onews`, and select Return all rows. Enter at least two records. For example, I entered the following:

onews_title	onews_body
My New blog!	This is my first blog post using my new blogging application.
Check out this new Apress book: Foundation ASP for Dreamweaver 8.	Make sure you pick up this new book on Dreamweaver 8 with ASP, available from friends of ED.

You'll notice that when you enter these records, the `onews_dtstamp` and `onews_online` columns will automatically populate due to the default values you added when creating the table.

Populating tbl_users

Now you'll add an administrator to this table. From the Tables section, right-click the name of your table, `tbl_users`, and select Return all rows. Enter at least one set of user credentials; for example:

username	user_pwd	user_level
omar	675fRo4Jg	admin

Make sure you manually change `user_level` to admin, as shown in Figure 9-8.

	user_id	username	user_pwd	dtstamp	user_level
▶	1	omar	123456	9/17/2005 1:39:55	admin
*					

Figure 9-8. View of actual record inserted in the tbl_users table

Setting user permissions for tables

Before closing Enterprise Manager, one more step you need to take is to create a SQL Server user who gets access to this database. No one will be able to make changes to this database without logging in as this user. You'll create a connection to this database in Dreamweaver through this user.

1. From the SQL Server console root, expand the Security folder, and right-click Logins from the menu.
2. Select New Login from the menu.
3. Under the General tab, type a name for your user in the Name field.

4. Select the radio button labeled SQL Server Authentication and type in a password. Specify the Blogs database as the default database for this user. Your dialog box should now look like Figure 9-9.

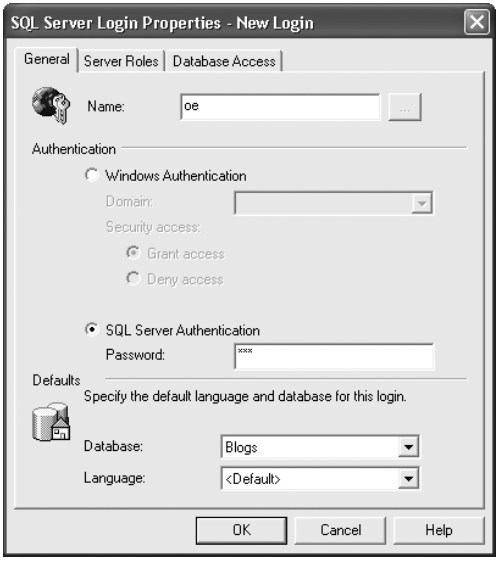


Figure 9-9. The General tab of the SQL Server Login Properties dialog box, on which you specify the user with login privileges

5. Now switch to the Database Access tab as shown in Figure 9-10, and permit this user access to the Blogs database by clicking the relevant check box.
6. Check the public and db_owner database roles; then click OK to close this dialog box.

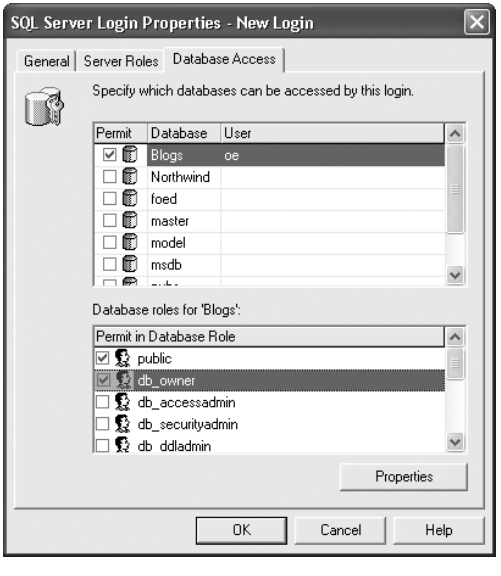


Figure 9-10. The Database Access tab of the SQL Server Login Properties dialog box, on which you can specify the database for which the user has privileges

Your database is now complete, and you can close Enterprise Manager. However, make sure SQL Server is still running.

> *Now that you've completed your database, you'll begin working in Dreamweaver. Before you start the next sections, you should define a new site named* `FOED_Chapter9`*, on which you'll view your dynamic ASP pages locally (or you can simply add a new folder named* `FOED_Chapter9` *to the site you defined earlier in this book. For a refresher on getting the environment ready, please see Chapter 2.*

Connecting Dreamweaver to your database

Before you begin creating the blog pages, you need to create a connection to the database in Dreamweaver. Dreamweaver will then create the ASP code for a database connection and then insert it into all the dynamic pages you specify.

1. Fire up Dreamweaver and select Databases ➤ + ➤ Custom Connection String from the Application panel (as shown in Figure 9-11). You can use either an ODBC or OLE DB connection string, or simply create a DSN (data source name). In this example, you'll use an ODBC connection string.
2. In the Custom Connection String dialog box, name your connection whatever you like (some recommendations are `connsql_foed` or `connaccess_foed`), and enter the following string, being sure to change the machine name, username, and password to your own:

   ```
   Driver={SQL Server};Server= MachineName;Database=Blogs;
   ➥ Uid=oe;Pwd=123;
   ```

3. Select Using Driver on this machine from the Custom Connection String dialog box.
4. Test your connection, and you should see the message Connection was made successfully.

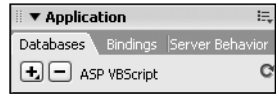


Figure 9-11. The Databases tab of the Application panel

If you received an error, it's because Dreamweaver can't connect to the database. This is likely because of one of the following three reasons:

- SQL Server is not running.
- Permissions for the database are not set properly.
- The username and password entered in the string does not match the SQL Server login.

To resolve these issues, review the "Setting user permissions for tables" section earlier in this chapter.

Connection strings

You can use either an ODBC or an OLE DB connection string. Here are the strings, customized for SQL Server.

ODBC:

```
Driver={SQL Server};Server= MachineName;
➥ Database=dbname;Uid=sa;Pwd=pass;
```

OLE DB:

```
Provider=SQLOLEDB;Data Source= MachineName;
➥ Initial Catalog= dbname;User ID=oe;Password=123;
```

Here are the strings, customized for Access.

ODBC:

```
Driver={Microsoft Access Driver (*.mdb)};
➥ Dbq=C:\somepath\databasename.mdb;
```

OLEDB:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
➥ C:\somepath\databasename.mdb;
```

DSN

You can also create a DSN, which will link your server to the database directly without your having to use a string.

1. Go to Start ➤ Control Panel ➤ Administrative Tools ➤ Data Sources (ODBC). From the System DSN tab, click the ADD button.
2. Select the SQL Server driver.
3. Enter a name for your DSN and select your server from the Server drop-down list.
4. Select Use SQL Server authentication, enter the username and password you created for yourself, click Next, and then click Finish.
5. Test your data source and then click OK. This DSN will then be available in Dreamweaver when you create your connection.

Be aware that if you go with a DSN, you'll need to have your web host create a DSN for you when you decide to upload your local pages to your remote server, which may or may not be an inconvenience. Also, most web hosts allow a minimal number of DSNs to be created, which will limit the number of database connections you can have if you want to add more databases to your website in the future. When you use a connection string, you don't need to create a DSN, since you're manually creating the connection string, but you'll need the SQL Server name and the username and password for your database on the server. Using a connection string also makes it easier to port your site over to a new web host, because you can just use the same DSN without fear of your data connection breaking.

Now that you have a successful connection between Dreamweaver and the database, you can proceed to creating your dynamic pages.

Displaying blog entries

In this section, you'll begin creating the Dreamweaver pages that will display the blog entries to your web users. Fire up Dreamweaver and get started! Remember that this section will contain three pages—let's go ahead and build your first page.

Creating onews_main.asp

This page will retrieve and display the blogs from your Blogs database that were created on the current day. It will display the titles for the blogs; each title will be hyperlinked to a second page that will display the individual blog details.

1. In Dreamweaver, create a new dynamic ASP VBScript page. Save it as onews_main.asp.
2. You'll first create the recordset that will retrieve the blogs for the current day. Select Application Panel ➤ Server Behaviors ➤ Recordset (Query) to open the Recordset dialog box.
3. Switch to Advanced mode. Although this is your first recordset, it will be a bit more complicated than the rest because it needs to display the records that were inserted for the current day.
4. Select the appropriate connection to your database, and name your recordset rsgettitles.
5. Enter the following SQL into the appropriate text field:

```
SELECT onews_id, onews_title, onews_dtstamp, onews_body
FROM dbo.tbl_onews
WHERE onews_dtstamp >= dateadd(day,datediff(day,0,GETDATE()), 0)
➥ AND onews_dtstamp <
➥ dateadd(day,1,dateadd(day,datediff(day,0,GETDATE()),
➥ 0)) AND onews_online = 1
```

Your Recordset dialog box should now look like Figure 9-12.

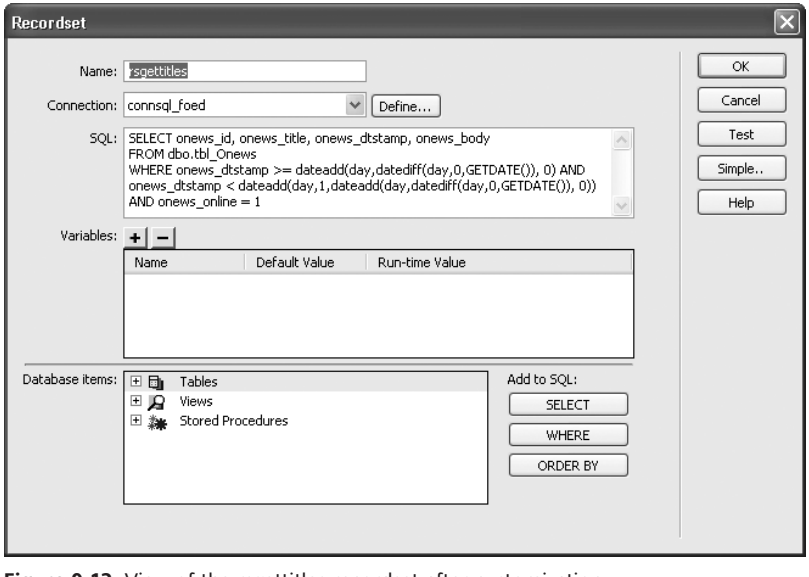


Figure 9-12. View of the rsgettitles recordset after customization

If you will be using Access, you will need to use the following code in place of the SQL above:

```
SELECT onews_id, onews_title, onews_dtstamp, onews_body
FROM tbl_Onews
WHERE onews_dtstamp >= dateadd(""d"",datediff(""d"",0,NOW()), 0)
➥ AND onews_dtstamp < dateadd(""d"",1,dateadd(""d"",datediff(""d"",
➥ 0,NOW()), 0)) AND onews_online = -1
```

Note that running a Dreamweaver Test through the dialog box may throw an error due to such a complicated SQL statement, but you should go ahead and click OK.

6. Test the recordset and you should see the results in a new Dreamweaver dialog box, as shown in Figure 9-13. If it's running OK, click OK to save the recordset; otherwise check the details you entered and try again.

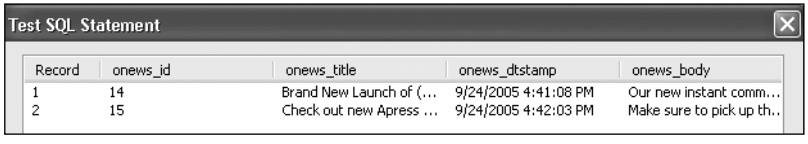


Figure 9-13. View of the rsgettitles recordset during a test

Once your recordset is created, the rest of the page is merely a matter of design. The page is currently retrieving the records but not displaying them—let's see to that next.

> *Notice that the names of SQL Server database tables need to be prefixed with* dbo. *Here is an example of how a SQL database table will be referred to within a SQL command:* `dbo.tbl_photoalbums` *and* `dbo.tbl_photos`. *However, in Access, dbo is not needed. In a SQL command connecting to an Access database, those tables will be referred to as simply* `tbl_photoalbums` *and* `tbl_photos`.

Designing onews_main.asp

1. In Dreamweaver, switch to Design view and create a header or insert an image to serve as the header of your page, as shown in Figure 9-14. You can use my example image if you like—it's available in the code download as `onews.gif`. It's a good idea to use the same header on all pages, for consistency's sake.

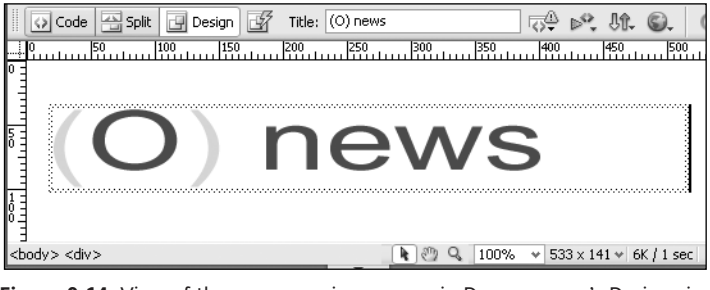


Figure 9-14. View of the onews_main.asp page in Dreamweaver's Design view

2. Insert three tables stacked over each other. The first and second tables should have two rows and one column, and the third table should have one row and one column. Set the tables to each have a length of 500 pixels.
3. Insert the text Today's Blogs: DATE in the first row of the first table. (Note: in the following figures, words appearing in all caps are placeholders that will be replaced with dynamic text.)
4. Insert the text TITLE in the second row of the first table.
5. Insert the text (Archives) (Administration) in the second row of the second table.
6. Link the Archives text to a page named `onews_archives.asp`.
7. Link the Administration text to a page named `onews_admin_archives.asp`, which you'll create later in this section.
8. In the third table, insert the following text: There is no news posted for today yet.

After following these steps, your page should now look like that shown in Figure 9-15.

Figure 9-15. The onews_main.asp page in Dreamweaver's Design view, after its design customization

Wiring up onews_main.asp

1. Now that you've designed your page, you're going to add the necessary code to it to make it work.
2. Highlight the text DATE from your web page, and then go to Application Panel ➤ Bindings, as shown in Figure 9-16.
3. Expand the `rsgettitles` recordset and select the `onews_dtstamp` column.
4. Click Insert from the bottom of the Application panel. The dynamic date of the record now replaces the static DATE text you previously had.

Figure 9-16. View of the Application panel's Bindings tab for the rsgettitles recordset

5. Next, highlight the text TITLE.
6. From the Bindings tab, select the onews_title column, and then click the Insert button. The dynamic title of the record now replaces the static TITLE text you previously had.
7. Highlight the row that includes the dynamic title (the second row of the first table).
8. Go to Application Panel ➤ Server Behaviors ➤ Repeat Region. Select rsgettitles from the Recordset drop-down list, check the Show: All records radio button to show all records (see Figure 9-17), and click OK. This will ensure that all titles in the recordset are listed.

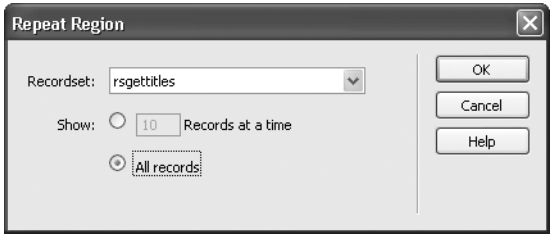


Figure 9-17. View of the rsgettitles Repeat Region behavior after its customization

Next, you'll ensure that these tables appear only when the recordset returns records, using the Show Region behavior.

9. Highlight both the first and second tables.
10. Go to Application Panel ➤ Server Behaviors ➤ Show Region ➤ Show Region If Recordset Is Not Empty.
11. When the dialog box opens, select the rsgettitles recordset (see Figure 9-18) and click OK. This will show the tables if records are returned.

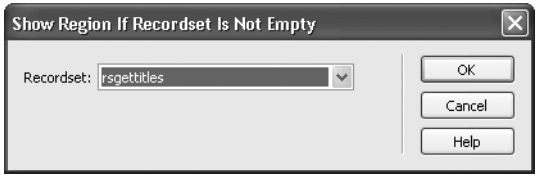


Figure 9-18. View of the Show Region behavior after its customization

12. Next, highlight the third table.
13. Go to Application Panel ➤ Server Behaviors ➤ Show Region ➤ Show Region If Recordset Is Empty.
14. When the dialog box opens, select the rsgettitles recordset and click OK. This will show this table with your message if records are not returned.

15. Finally, you'll link the titles to the details page, which you'll create next. Highlight the dynamic title in Dreamweaver, go to Link under the Properties menu, and select the yellow folder to create your link.
16. When the dialog box opens, type onews_details.asp for the file name, and then select the Parameters button.
17. In the Parameters dialog box, type onid for the name of the parameter and select the yellow lightning bolt to select the value.
18. Select the onews_id column from the recordset and click OK (see Figure 9-19); then click OK twice more to close the dialog boxes. Figure 9-20 shows the Parameters dialog box after the parameter value has been selected.

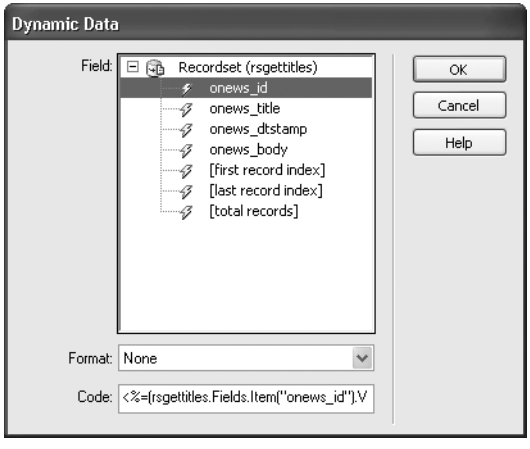


Figure 9-19. View of the Dynamic Data dialog box for the rsgettitles recordset

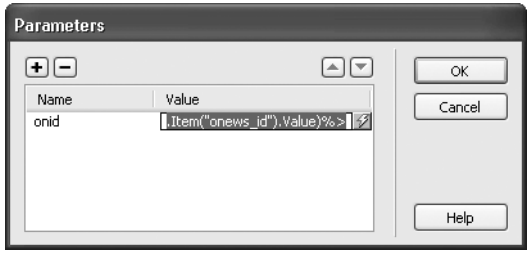


Figure 9-20. View of the Parameters dialog box after the URL parameter, onid, has been added

Here, you've created a hyperlink for each title. Each title passes the unique ID of the clicked blog entry to the `onews_details.asp` page, which you'll build next. The correct information for that blog entry is then retrieved and displayed. In Design view, your `onews_main.asp` page should now look like that shown in Figure 9-21.

Figure 9-21. View of onews_main.asp in Design view after dynamic content has been added

Now test your page—you should now be able to view the page in your browser; it will look something like Figure 9-22.

(O) news - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
Today's Blogs: Saturday, September 24, 2005
Brand New Launch of (O) news!
Check out new Apress book: Foundation ASP for Dreamweaver 8
(Archives) (Administration)
Local intranet

Figure 9-22. View of onews_main.asp in the web browser

Creating onews_details.asp

As discussed before, this page will display the full details of the blog entry selected from the onews_main.asp page. It will display the date, title, and body text of the blog entry. Let's build it now.

1. In Dreamweaver, create a new dynamic ASP VBScript page and save it as onews_details.asp.

   How does this page know which blog was selected from the previous page? Because the unique ID was passed into the URL as a URL parameter. So now all you have to do is find the record whose ID matches the URL parameters that you labeled onid. Next, you'll create a recordset that will retrieve the blog selected from the previous page.

2. Select Application Panel ➤ Server Behaviors ➤ Recordset (Query).
3. Name your recordset rs_getblog.
4. Select your connection and the table labeled tbl_onews.
5. Select all database columns except the onews_online column.
6. Filter the onews_id column, making sure it's equal to the URL parameter labeled onid. Your Recordset dialog box will now look like Figure 9-23.

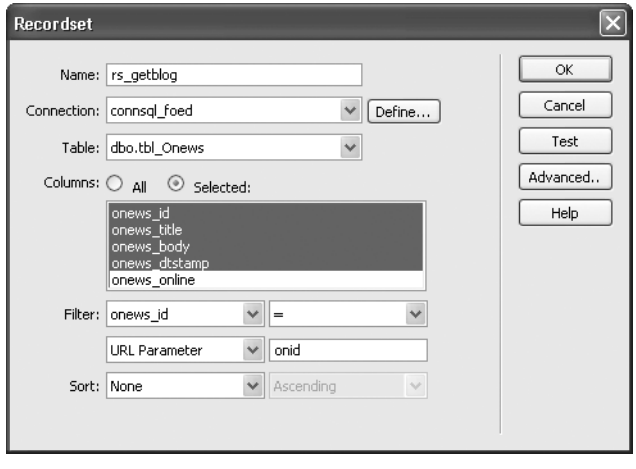


Figure 9-23. View of the rs_getblog recordset after customization

7. Switch to Advanced view. You need to do this because you need to add a second condition—Simple mode can only use one condition in the SQL WHERE command. After the WHERE command, make sure to add the SQL code highlighted in Figure 9-24.

```
SELECT onews_id, onews_title, onews_body, onews_dtstamp
FROM dbo.tbl_Onews
WHERE onews_id = MMColParam AND onews_online = 1
```

Figure 9-24. View of SQL in Advanced mode

This makes sure that only records that have a onews_online value of 1 are displayed. Those that have a value of 0 will not display; this can be set for each blog entry using the administrative controls you'll create later.

For Access, replace the SQL from Figure 9-24 with the following SQL:

```
SELECT onews_id, onews_title, onews_body, onews_dtstamp
FROM tbl_Onews
WHERE onews_id = MMColParam AND onews_online = -1
```

Designing onews_details.asp

1. In Dreamweaver, switch to Design view and create the title of your page as you did with your first page.
2. Link this title or image to onews_main.asp so the visitor has an easy way to return to the main page.
3. After the title, insert two tables—the first table should have three rows, and the second table should have one.
4. In the first row of the first table, type the text TITLE.
5. In the second row of the first table, type the text DATE.
6. In the third row of the first table, type the text BODY.
7. In the second table, type the text Ooops. Item has either been deleted or taken offline.

Your page should now look like that shown in Figure 9-25.

Figure 9-25. View of onews_details.asp in Dreamweaver before adding dynamic behaviors

Wiring up onews_details.asp

1. Highlight the text TITLE from your web page, and then go to Application Panel ➤ Bindings.
2. Expand the rs_getblog recordset and select the onews_title column. Click Insert from the bottom of the Application panel. The dynamic title of the record now replaces the static TITLE text you had.

3. Replace the DATE text with the dynamic date from the recordset.
4. Replace the BODY text with the dynamic body from the recordset.
5. Highlight the first table and apply a Show Region If Recordset Is Not Empty behavior.
6. Finally, highlight the second table and apply a Show Region If Recordset Is Empty behavior. You page should now look like Figure 9-26.

Figure 9-26. View of onews_details.asp in Dreamweaver after adding dynamic behaviors

Your `onews_details.asp` page is complete. You should test this page by going to the `onews_main.asp` page first. Try selecting a blog title in order to view the details page—it should look something like Figure 9-27.

(O) news - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
Brand New Launch of (O) news!
Saturday, September 24, 2005
Our new instant communication section has been launched. It's called O news. This is where you will find me directly communicating with you. I will give you the latest, most important news you need to know about our company. Make sure you check O news everyday to make sure you aren't missing anything that could definitely involve you.
Local intranet

Figure 9-27. View of the onews_details.asp page in the web browser

Creating onews_archives.asp

This page will retrieve all blogs in the database so your users can view past blogs. Let's get started without further ado.

1. In Dreamweaver, create a new dynamic ASP VBScript page, and save it as `onews_archives.asp`.

   The SQL for this recordset is simple because you simply need to retrieve all records—but make sure that only those meant to be online are displayed.

2. Go to Application Panel ➤ Server Behaviors ➤ Recordset (Query).
3. Name your recordset `rs_getblogs`.
4. Select your connection and the table labeled `tbl_onews`.
5. Select all database columns except the `onews_online` and `onews_body` columns; you don't need to display them in the page.
6. Filter the `onews_online` column, making sure it is equal to the Entered Value labeled 1. For Access, make sure the `Entered Value` is labeled -1.
7. Sort the `onews_dtstamp` column in descending order (your Recordset dialog box should now look like Figure 9-28), and click OK to finish.

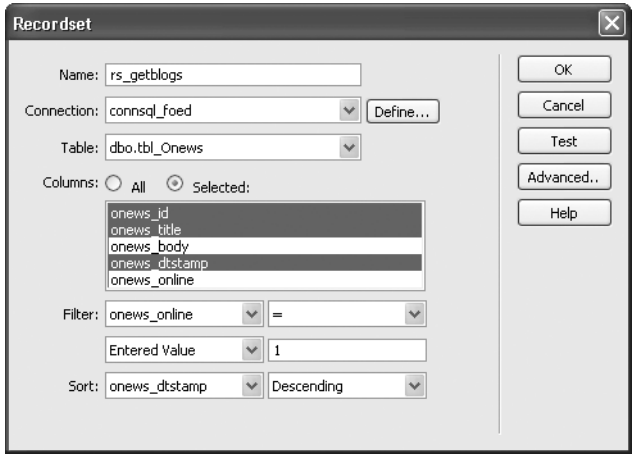


Figure 9-28. View of the rs_getblogs recordset after customization

Designing onews_archives.asp

1. In Dreamweaver, switch to Design view and create the title of your page as you did with your first page.
2. Link this title or image to `onews_main.asp` so the visitor has an easy way to return to the main page.
3. After the title, insert two tables. The first table should have two rows, and the second table one row.
4. The first row of the first table should have two columns. In the left column of the first row, type the text Main - Archives.

5. Link the Main text to onews_main.asp. Leave the right column of the first row empty for now.
6. In the second row, type the text DATE - TITLE.
7. The second table should have one row. In this row, type the text There is no news posted yet.

Your page should now look like that shown in Figure 9-29.

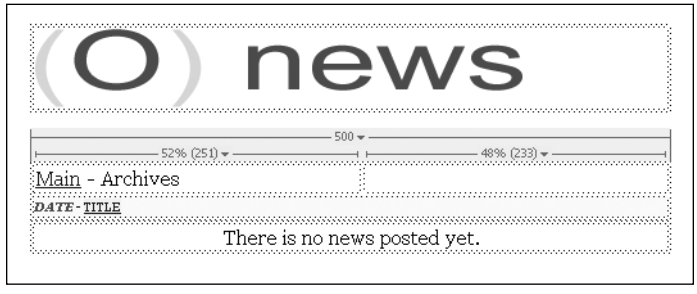


Figure 9-29. View of onews_archives.asp in Dreamweaver's Design view before adding dynamic behaviors

Wiring up onews_archives.asp

1. Highlight the DATE text from your web page.
2. Select Application Panel ➤ Bindings.
3. Expand the recordset (rs_getblogs) and select the onews_dtstamp column.
4. Click Insert at the bottom of the Application panel. The dynamic date of the record now replaces the static DATE text you previously had.
5. Replace the TITLE text with the dynamic title from the recordset.
6. Highlight the second row in the first table and apply a Repeat Region behavior—make sure you select the rs_getblogs recordset. Click OK.
7. Highlight the second table and apply a Show Region If Recordset Is Empty behavior. Select the rs_getblogs recordset and click OK. Your page should now look like Figure 9-30.

Figure 9-30. View of onews_archives.asp in Dreamweaver's Design view after adding dynamic behaviors

8. Finally, highlight the dynamic title and link it back to the onews_details.asp page using the Properties panel, passing the onews_id column (from the rs_getblogs recordset) as a URL parameter labeled onid, just like you did in the onews_main.asp page.

Now view this page in your browser by selecting the Archives hyperlink from the onews_main.asp page. All blogs should be listed in descending order by date, as shown in Figure 9-31.

(0) news Archives - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
Main - Archives
September 24, 2005 - Check out new Apress book: Foundation ASP for Dreamweaver 8
September 24, 2005 - Brand New Launch of (O) news!
Done
Local intranet

Figure 9-31. View of onews_archives.asp in the web browser

You've now completed the first major component of your blog. Next, you'll create the administration controls, which will allow you to insert new blogs and update current blogs.

Blog administration

The administration controls span only two pages. The first page (onews_admin_archives.asp) will allow you to insert new blog entries and view the blog archive, while the second page (onews_admin_details.asp) will give you the ability to update or delete selected blog entries from the first page.

Before you start creating the pages, create a new folder labeled admin within the root of your website. Place the administration control pages in this folder.

Creating onews_admin_archives.asp

1. In Dreamweaver, create a new dynamic ASP VBScript page and save it as onews_admin_archives.asp.
2. Go to Application Panel ➤ Server Behaviors ➤ Recordset (Query).
3. Name your recordset rs_getblogs.

4. Select your connection and the table labeled `tbl_onews`.
5. Select all database columns except the `onews_body` column.
6. Order the records by the `onews_dtstamp` column in descending order.

This recordset retrieves all blogs in descending order by date of creation; the final Recordset dialog box should look like Figure 9-32. If it's correctly filled in, click OK.

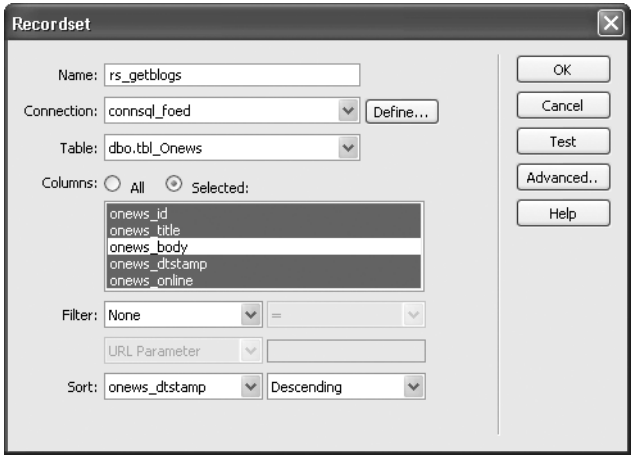


Figure 9-32. View of the rs_getblogs recordset after its customization

Designing onews_admin_archives.asp

1. In Dreamweaver, switch to Design view and create the title of your page as you did in your first page.
2. Link this title or image to `onews_main.asp` so you have an easy way to return to the main page.
3. After the title, insert one table with five rows.
4. Split the first row to have two columns.
   a. In the left column of the first row, type the text Insert/Update Blog Post.
   b. In the right column of the first row, type the text Administration Archives.
5. In the second row, insert a form named `frm_insert`.
   - Within the form, insert a text field with the name `txt_title`, and set it to have a character width of 60.
   - Insert a line break and insert a text area named `txt_body` with dimensions of 45 characters by 5 lines. Make sure the text area is set to Multi line.
   - Insert a line break; then insert a form button named `Submit` and give it a label of Submit (O) news (or something similar, like Submit Blog Post).
6. In the fourth row, insert the text DATE -TITLE- (update) -.

Your page should now look like that shown in Figure 9-33.

Figure 9-33. View of onews_admin_archives.asp in Dreamweaver's Design view before adding dynamic behaviors

Wiring up onews_admin_archives.asp

1. Highlight the DATE text from your web page, and then go to Application Panel ➤ Bindings.
2. Expand the recordset (`rs_getblogs`) and select the `onews_dtstamp` column.
3. Click Insert from the bottom of the Application panel. The dynamic date of the record now replaces the static DATE text you previously had.
4. Replace the TITLE text with the dynamic title from the `rs_getblogs` recordset from the Bindings tab.
5. Link the (update) text to the `onews_admin_details.asp` page. Make sure you pass the `onews_id` of the `rs_getblogs` as a URL parameter named `onid`, as you did in previous pages.
6. Next, you need to create the dynamic insert. Go to Application Panel ➤ Server Behaviors ➤ + ➤ Insert Record to open the Insert Record dialog box.
7. Fill in the Insert Record dialog box as detailed in the following substeps (and shown in Figure 9-34).
   a. Select your connection.
   b. Insert into the table labeled `dbo.tbl_onews`.
   c. Set the page to come back to the same page (`onews_admin_archives.asp`) after inserting.
   d. Get the values from the form named `frm_insert`; set `txt_title` to insert into `onews_title` as text, and set `txt_body` to insert into `onews_body` as text also.

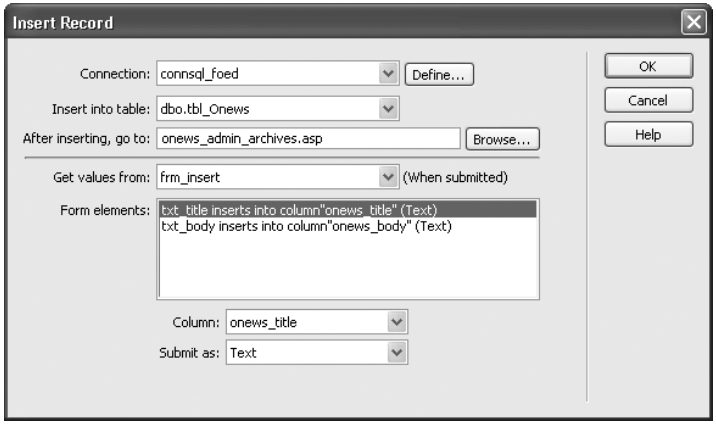


Figure 9-34. View of Insert Record behavior after customization

8. Check your dialog box, and if everything is correct, click OK.

   Finally, you'll display a message letting you know which blogs are online and which ones are offline.

9. In Design view, place your cursor to the right of the hyphen on the right side of the (update) text.

10. Switch to Code view and insert the following code:

```
<% If (rs_getblogs.Fields.Item("onews_online").Value) = True Then
➥ Response.Write "Live" Else Response.Write "Offline" End If %>
```

    This will write the word Live next to all blog entries whose onews_online columns have values of 1, and Offline next to all blog entries whose onews_online columns have values of 0.

Your page is now complete—the form will now allow you to insert a blog entry into the tbl_onews database table. In Design view, it should currently look like Figure 9-35.

Figure 9-35. View of onews_admin_archives.asp in Design view after adding dynamic behaviors

Now test this page by loading it in your browser and attempting to insert a blog—your finished web page will look like that shown in Figure 9-36.

(O) news Administration Archives - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
Insert/Update (O) news
Administration Archives
Submit (O) news
September 24, 2005 - Check out new Apress book: Foundation ASP for Dreamweaver 8 - (update) - Live
September 24, 2005 - Brand New Launch of (O) news! - (update) - Live
Local intranet

Figure 9-36. View of onews_admin_archives.asp in the web browser

Your insert page is complete; now you need to create a page giving you the ability to update blogs.

Creating onews_admin_details.asp

This page will have three functions—to update a blog, to delete a blog, and to change a blog's online status. Let's build it now.

1. In Dreamweaver, create a new dynamic ASP VBScript page and save it as `onews_admin_details.asp`.
2. Go to Application Panel ➤ Server Behaviors ➤ Recordset (Query) to open up the Recordset dialog box.
3. Fill in the dialog box as shown in Figure 9-37. Here, you'll retrieve the individual blog selected from the previous page (`onews_admin_archives.asp`) in order to perform an administrative update action. Remember that, on the previous page, you passed `onews_id` as a value in a URL parameter named `onid`.
   a. Name your recordset `rs_getblog`.
   b. Select your connection and the table labeled `tbl_onews`.
   c. Select all database columns.
   d. Filter the records, making sure the `onews_id` is equal to the URL parameter named `onid`.
   e. Click OK.

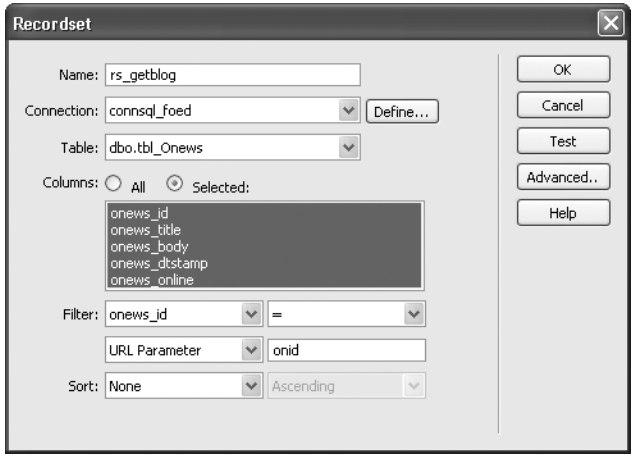


Figure 9-37. View of the rs_getblog recordset after customization

Designing onews_admin_details.asp

1. In Dreamweaver, switch to Design view and create the title of your page as you did on your first page.
2. Link this title or image to `onews_main.asp` so you have an easy way to return to the main page.
3. After the title, insert a table with seven rows and one column.
4. Split the first row into two columns.
5. In the left column of the first row, type the text Update Blog Post details.
6. In the right column of the first row, type the text Administration Home, and link that text to the `onews_admin_archives.asp` page.
7. In the third row, insert a form named `frm_update`. Within the form, do the following:
   a. Insert a text field named `txt_title` and set it to have a character width of 60.
   b. Insert a line break, and insert a text area named `txt_body` with dimensions of 45 characters by 5 lines. Make sure the text area is set to Multi line.
   c. Insert a line break, and insert a check box named `chkb_onlinestatus`. Type the following text to the right of the check box: Online Status (checked = live, unchecked = Offline).
   d. Insert a line break, and then insert a text field named `txt_dtstamp`. Set it to have a character width of 20.
   e. Highlight this text field and switch to Code view. Add the following code to the before the end of the HTML tag:

```
<input name="txt_dtstamp" type="text" id="txt_dtstamp"
➥ size="20" disabled="true" />
```

   f. Insert two line breaks then insert a form button named `Submit` with a label of Update Blog Post.

8. In the fifth row, insert the text Delete Record Forever.
9. In the sixth row, insert a form named `frm_delete`. Within this form, do the following:
   a. Insert a form button named `submit_delete`, with a label of Delete Record Forever.
   b. Highlight this button and switch to Code view; then edit the HTML tag, adding the following bold code:

```
<input type="submit" name="submit_delete"
value="Delete Record Forever" disabled="true" />
```

   c. Type the following text to the right of the `submit_delete` button: Click here to enable delete button.
   d. Highlight this text, and from the Properties panel, link the text to the following:

```
JavaScript:enableDelete()
```

10. Switch to Code view and scroll up to the HTML <head> tag. Insert the following JavaScript before the closing </head> tag:

```
<script language="JavaScript">
function enableDelete()
{
  document.frm_delete.submit_delete.disabled=false;
}
</script>
```

This JavaScript function adds an extra stage of fool-proofing to your page. When this page is viewed in the browser, you'll have to enable the delete button before you can actually submit the delete form. This is so the delete button is not accidentally submitted.

11. Create a second table with only one row and column just below the first table; add the text Item does not exist to it, as shown in Figure 9-38.

Figure 9-38. View of onews_admin_details.asp in Design view before dynamic behaviors have been added

You're now done with the design aspect of the update page. Now you need to add the dynamic aspects to wire-up the page functionality.

Wiring up onews_admin_details.asp

It's time to create the update functionality—fortunately for you, Dreamweaver makes this very easy!

1. Select Application Panel ➤ Server Behaviors ➤ + ➤ Update Record to open the Update Record dialog box, as shown in Figure 9-39.

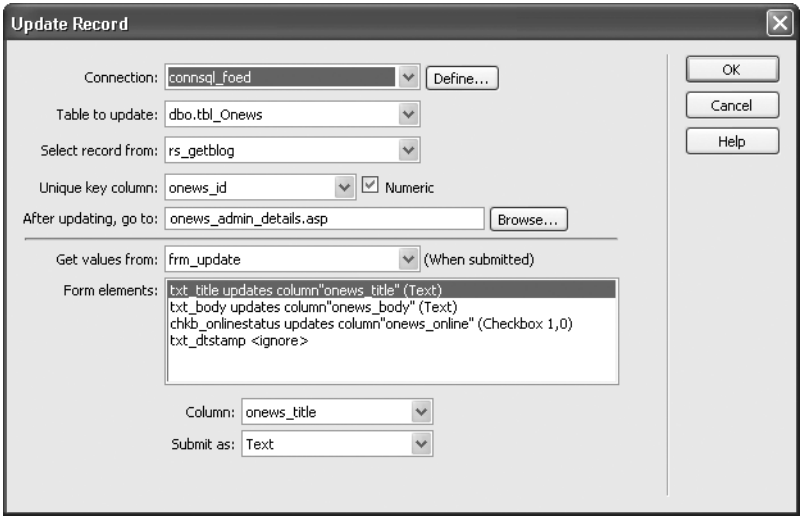


Figure 9-39. View of Update Record behavior after customization

2. Select your connection.
3. Select the `dbo.tbl_onews` as the table to update.
4. Select the record from the `rs_getblog` recordset.
5. Use `onews_id` as the Unique key column. Make sure Numeric is checked.
6. Set the After updating, go to field so that you're redirected back to the same page, `onews_admin_details.asp`.
7. Set the behavior to get values from the `frm_update` form.
8. For the elements, make sure `txt_title` updates the `onews_title` column as `Text`, `txt_body` updates the `onews_body` column as `Text`, and `chkb_onlinestatus` updates the `onews_online` column as `(Checkbox 1,0)`.
9. Just set `txt_dtstamp` to be ignored. You don't want to update the date of creation; you just want to display it in the form for reference.
10. Click OK to save the behavior.

    Now, whenever the update form is submitted, it will update the appropriate columns in the database. You must now make sure that the form is populated with the current column values, too, so that you can view what is already in the record and won't overwrite the record with empty values! You can achieve this by binding the update form with the current database column values.
11. In Design view, highlight the `txt_title` text field in the update form.

12. Select Application Panel ➤ Bindings.
13. Expand the rs_getblog recordset and select the appropriate database column, onews_title.
14. On the Bindings tab, click the Bind button. Bind the rest of the database columns in the rs_getblog recordset from the Bindings tab to their appropriate form fields on the page, as specified in the following substeps:
    a. Highlight the txt_body text area in the update form and bind the onews_body column to it.
    b. Highlight the chkb_onlinestatus check box in the update form and bind the onews_online column to it.
    c. Highlight the txt_dtstamp text field in the update form and bind the onews_dtstamp column to it.

    The update form is now complete. Now you'll jump to the delete form and add the behavior that will enable deletion of the currently viewed blog entry.
15. Select Application Panel ➤ Server Behaviors ➤ + ➤ Delete Record to open up the Delete Record dialog box, as shown in Figure 9-40.

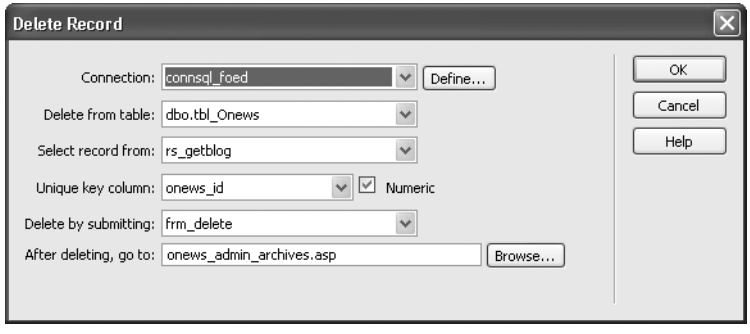


Figure 9-40. View of the filled-in Delete Record dialog box

16. In this dialog box, first select your connection.
17. In the Delete from table field, select dbo.tbl_onews.
18. Select the record from the rs_getblog recordset.
19. Use the onews_id as the Unique key column. Make sure Numeric is checked.
20. Set the value of Delete by submitting to frm_delete.
21. In the After deleting, go to field, select the onews_admin_archives.asp page.
22. Click OK.

You can now delete individual blog entries using this form.

Finally, you'll use the Show Region behavior to show the blog only if it actually exists. Even though the blog is being selected from a previous page that lists records directly from the database, a site visitor can play with the querystring manually and enter an ID for a blog that does not exist. Instead of allowing your web application to be thrown off due to the request for a nonexistent database record, you cover yourself by using the Show Region behavior. Select the first table and add the Show Region If

Recordset Is Not Empty behavior. Then select the second table and add the Show Region If Recordset Is Empty behavior. In Design view, your page will now look like Figure 9-41.

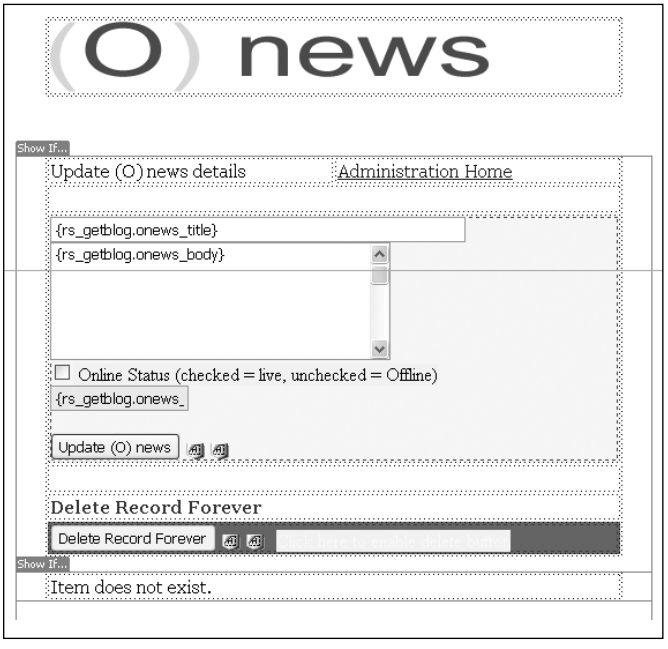


Figure 9-41. View of onews_admin_details.asp in Design view after dynamic behaviors have been added

Now view the page live in your web browser. Start out by surfing to the `onews_admin_archives.asp` page. Select a record to update, and test the update, as shown in Figure 9-42.

(O) news
Update (O) news details
Administration Home
Brand New Launch of (O) news!
Our new instant communication section has been launched. It's called O news. This is where you will find me directly communicating with you. I will give you the latest, most important news you need to know about our company. Make sure you check O news everyday
Online Status (checked = live, unchecked = Offline)
9/24/2005 4:41:08 PM
Update (O) news
Delete Record Forever
Delete Record Forever
Click here to enable delete button

Figure 9-42. View of onews_admin_details.asp in the web browser upon selection of a particular blog entry

Test out updating the text, changing status from online to offline, and deleting entries.

Congratulations! You have completed the bulk of your blog application. Now the only tasks left are securing the administration pages, creating the administrator login/logout functionality, and creating the search feature. You'll look at the first two in the next section, as they go hand in hand.

Securing the blog administration pages and creating login/logout functionality

Currently, your administration pages can be accessed by anyone, since you've done nothing to secure access to them. Let's secure them now.

1. In Dreamweaver, open both `onews_admin_archives.asp` and `onews_admin_details.asp`.
2. Starting with `onews_admin_archives.asp`, select Application Panel ➤ Server Behaviors ➤ + ➤ User Authentication ➤ Restrict Access to page to open up the Restrict Access To Page dialog box, as shown in Figure 9-43.

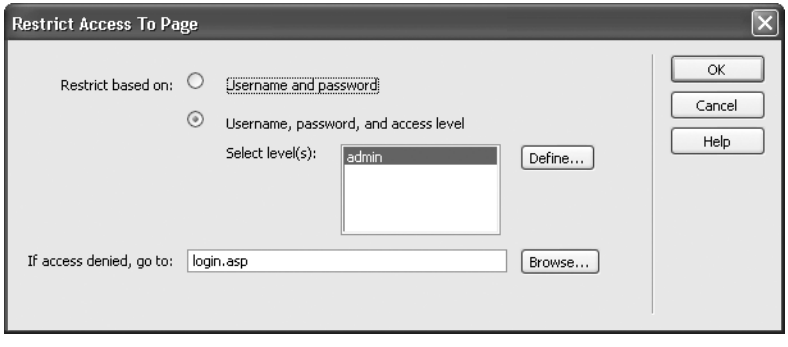


Figure 9-43. View of the Restrict Access To Page dialog box after customization

3. Check the Restrict based on: Username, password, and access level radio button.

   Remember that in your database you created a column named `user_level`, in which you gave your administrator a value of `admin`. Here, you need to define the `admin` level as being allowed access, since this is the only level of user you want to allow access to the administration pages.
4. In the If access denied; go to field, choose the `login.asp` page (you'll create this page next).
5. Click OK, then complete these same steps for the `onews_admin_details.asp` page.

Dreamweaver will make sure that a session created by the `login.asp` page has the value `admin`. If not, it will disallow access and send the user back to the login page.

Now you need to create the login page that will create the necessary ASP sessions. You'll no longer be able to access any of the administration pages unless you log in.

Designing login.asp

1. Create a new dynamic ASP VBScript page and save it as `login.asp`.
2. In Design view, create the title of your page as you did for your first page.
3. Link this title or image to o`news_main.asp` so you have an easy way to return to the main page.
4. Below your title, create a table with three rows and one column.
5. Split the first row to have two columns.
6. In the left column of the first row, type the text Login to Administrate.
7. In the third row, insert a form named `frm_login`. Within the form, do the following:
   a. Insert a text field named `txt_username`; to the right of the text field, type the text Username.
   b. Insert a line break, and then insert a text field named `txt_user_pwd`; to the right of the text field, type the text Password.
   c. Insert two line breaks, and insert a form button named `Submit` with a label of Login to (O) news (see Figure 9-44).

Figure 9-44. View of login.asp page in Design view

Wiring up login.asp

To save space here, you'll go through this quickly—making this page work merely involves using a Log In User server behavior. Select Application Panel ➤ Server Behaviors ➤ + ➤ User Authentication ➤ Log In User, and set the values of the fields as shown in Figure 9-45. Make sure the appropriate text fields validate through the appropriate database columns.

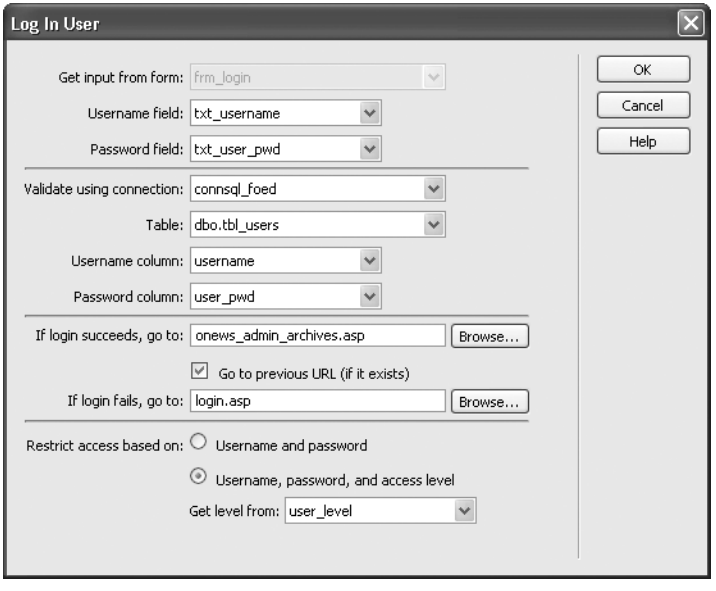


Figure 9-45. View of Log In User behavior after dialog box is filled in

What this does is create two sessions for the person who logs in. It creates one session named `MM_Username` with `username` as the value, and one session named `MM_UserAuthorization` with `user_level` as the value. So, when I log in, I will have a session with the value `Omar` and another session with the value `admin`.

Before you log in, display the value of these sessions on `onews_admin_archives.asp` so you can see that they're real. Open your `onews_admin_archives.asp` page, go to Application Panel ➤ Bindings ➤ + ➤ Session Variable, and enter `MM_Username` (see Figure 9-46).

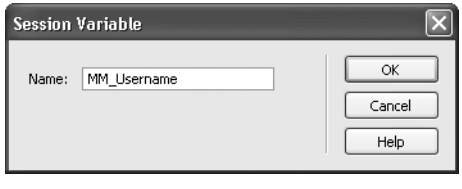


Figure 9-46. View of session variable dialog box, in which you can add the name of session variable to be made available on pages related to your site (simply by dragging and dropping from the Bindings tab of the Application panel)

Add another session variable, this time named `MM_UserAuthorization`, to the Bindings tab of the Application panel. You should now see the variables listed in the Bindings tab (see Figure 9-47).

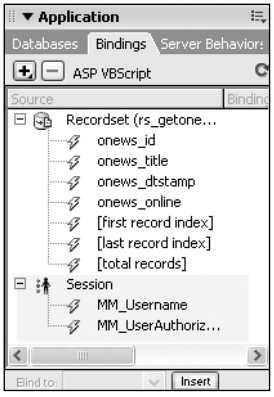


Figure 9-47. View of the Bindings tab with session variables included

Now drag and drop the session variable anywhere on your page. After doing this, try logging in to your `login.asp` page through your browser. When you login, you'll be redirected to the `onews_admin_archives.asp` page, and you'll see the value of the sessions displayed on your page.

Keep in mind that it is because of these sessions that you have access to the administration pages. Once these sessions are destroyed, you'll no longer have access to these pages.

The last thing to do in this section is to create a logout page so that you can close your sessions when you leave your computer, so that no one else can use the administration pages.

Creating logout.asp

This page has no HTML aspect; it's only meant to destroy the user sessions and redirect you back to the main page.

1. In Dreamweaver, create a new dynamic ASP VBScript page and save it as `logout.asp`.
2. Select Application Panel ➤ Server Behaviors ➤ + ➤ User Authentication ➤ Log Out User to open up the Log Out User dialog box, as shown in Figure 9-48.

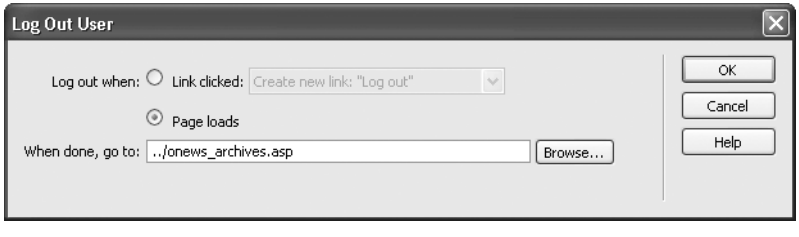


Figure 9-48. View of Log Out user behavior after customization

3. Set the server behavior to log out once the page loads and redirect to the main page, `onews_main.asp`.

Once this page is accessed, it will destroy the sessions and redirect to `onews_main.asp`. You'll no longer have access to the administration pages, and you'll need to log in again if you want access.

So, to add logout functionality to any page, you simply need to add the text Logout and link the text to the `logout.asp` page. That's it. Go ahead and try it, by adding such a link to your `onews_admin_archives.asp` page, next to where you dragged and dropped the user session variables (see Figure 9-49).

(O) news Administration Archives - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
omar - admin - Logout
Insert/Update (O) news
Administration Archives
Submit (O) news
September 24, 2005 - Check out new Apress book: Foundation ASP for Dreamweaver 8 - (update) - Live
September 24, 2005 - Brand New Launch of (O) news! - (update) - Live
Done
Local intranet

Figure 9-49. View of onews_admin_archives.asp in the web browser after adding session variables and the Logout link

Closing your browser or stalling 20 minutes with no server interaction will also destroy the session by default on the web server—this is the default amount of time the session will last. You can change the session timeout limit to 10 minutes by adding the following code to each page: `<% Session.Timeout=10 %>`*.You can also change this value for an entire application, rather than just one page, by opening* Internet Services Manager*, right-clicking* Default Web Site*, selecting* Properties*, going to the* Home Directory *tab of the application, clicking* Configuration*, selecting the* Options *tab, and altering the* Session Timeout *field.*

You are now done with securing your blog application. Go ahead and play with it for a little—adding, viewing, updating, and deleting blogs. Make sure you log in with the username and password of an administrator so that the administrative pages are accessible to you. If you forgot the username and password you specified, open the database table in SQL Server or Access to see the values for the record. In the next section, you'll create a nice little search feature for the blog.

Searching your blog

Your search feature will allow you and your users to search your blogs for keywords or phrases, by submitting them through a search form, in the same fashion as www.google.com. The first step is to create the search form.

Creating the search form

1. Open your onews_archives.asp page.

   When you first created this page, you left the right-hand column of the first table row empty. This space has been left for your search form, so you'll create it in this space.

2. In Design view, place your cursor in this table cell and insert a form named frm_search. Within the form, do the following:

   a. Insert a text field named txt_searchterm.

   b. To the right of this text field, insert a form button named Submit and give it a label of Search. Your page should now look like Figure 9-50.

Figure 9-50. View of onews_archives.asp in Dreamweaver, with the search form

3. Highlight the form and set its action to onews_searchresults.asp, which you'll create next.
4. Make sure the form method is set to Post.

You can actually place this search form on any of your pages. As long as you name the text field txt_searchterm, and you set the action to onews_searchresults.asp, it will send the query to the onews_searchresults.asp page.

Creating the results page

1. In Dreamweaver, create a new dynamic ASP VBScript page and save the page as onews_searchresults.asp.
2. Go to Application Panel ➤ Server Behaviors ➤ + ➤ Recordset (Query) to open the Recordset dialog box, and switch to Advanced mode.

3. Name your recordset rs_getblogs and select your connection. Enter the following SQL:

```
SELECT *
FROM dbo.tbl_onews
WHERE ' ' + onews_body + ' ' LIKE '%[^a-z]MMColParam[^a-z]%'
➥ OR ' ' + onews_title + ' ' LIKE '%[^a-z]MMColParam[^a-z]%'
➥ AND onews_online = 1
```

For Access, replace the SQL from above with the following SQL command:

```
SELECT *
FROM tbl_Onews
WHERE ' ' + onews_body + ' ' LIKE '%MMColParam%' OR ' ' +
onews_title + ' ' LIKE '%MMColParam%' AND onews_online = -1
```

4. Add the following to the Variables window:

Name	Default Value	Run-time Value
MMColParam	x	Request.Form("txt_searchterm")

Your Recordset dialog box should now look like Figure 9-51. The SQL searches both the body and title of the blog entry for the whole word or exact phrase entered into the search form on the previous page (onews_archives.asp).

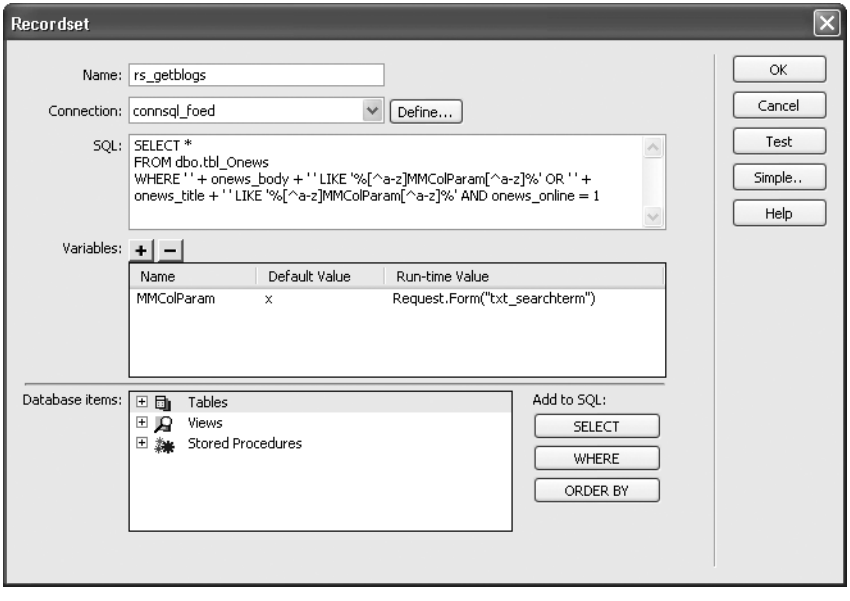


Figure 9-51. View of the rs_getblogs recordset after its completion

Designing onews_searchresults.asp

Now that you have the recordset sorted, you can design the page to actually display the results. This is very simple, so it won't take you long.

1. In Design view, create the title of your page as you did for your first page.
2. Link this title or image to `onews_main.asp` so you have an easy way to return to the main page.
3. Below your title, enter the text Search Results.
4. Below this text, insert two tables, one stacked over the other, with one row and one column each. In the first table, insert the text DATE - TITLE. In the second table, insert the text Your search returned no results.

Wiring up onews_searchresults.asp

1. Highlight the DATE text and bind the dynamic date from the `rs_getblogs` recordset in the Bindings panel to the DATE text.
2. Highlight the TITLE text and bind the dynamic title from the `rs_getblogs` recordset in the Bindings panel.

   Now you'll link your title to the `onews_details.asp` page by passing the value of the `onews_id` column as the value of a URL parameter named `onid`.
3. Highlight the title and select the folder icon next to the Link text in the Properties panel to open the Select File dialog box.
4. Enter `onews_details.asp` in the URL field, and click the Parameters button.
5. Add a parameter name `onid` and select the value as the `onews_id` column from the `rs_getblogs` recordset, as shown in Figure 9-52.

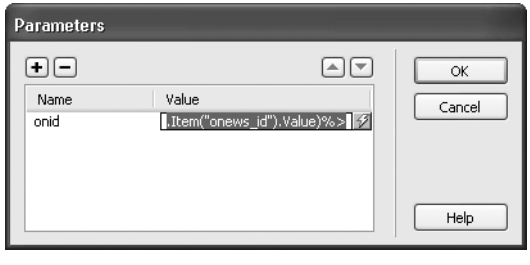


Figure 9-52. View of the Parameters dialog box after having added the onid URL parameter

6. Highlight the first row in the first table, and add a Repeat Region behavior using the `rs_getblogs` recordset. Make sure you show all records, and then click OK.
7. Highlight the second table and add the Show Region If Recordset Is Empty behavior to it. This will display the message Your search returned no results. if the recordset comes back empty (meaning the keyword was not found).

Your search feature is now complete. Test it by loading your `onews_archives.asp` page in your web browser and searching for a term—first, search for a term that doesn't exist, and you'll get the output shown in Figure 9-53.

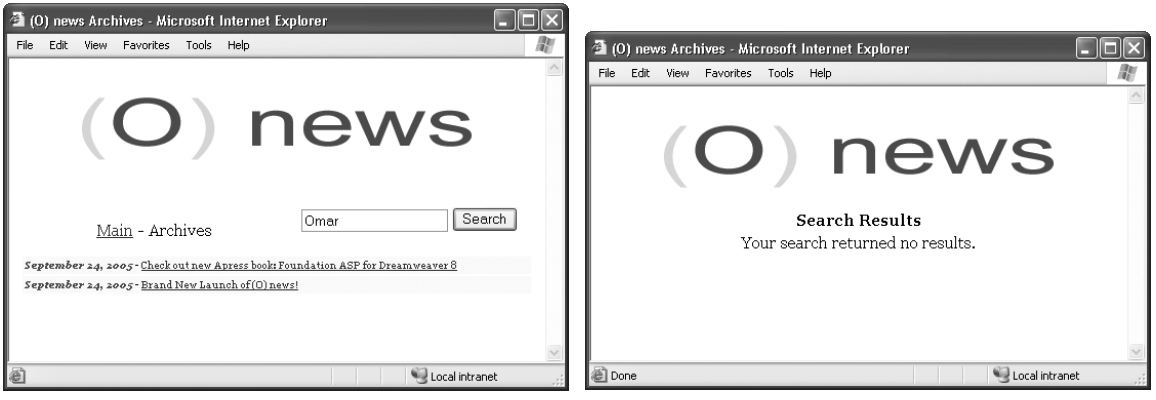

(0) news Archives - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
Main - Archives
Omar
Search
September 24, 2005 - Check out new Apress book: Foundation ASP for Dreamweaver 8
September 24, 2005 - Brand New Launch of (O) news!
Local intranet
(0) news Archives - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
Search Results
Your search returned no results.
Done
Local intranet

Figure 9-53. Searching for a keyword that doesn't exist in your blog entries

Now try searching for a term you know exists in the title or body of a blog entry, and you'll get output like that shown in Figure 9-54.

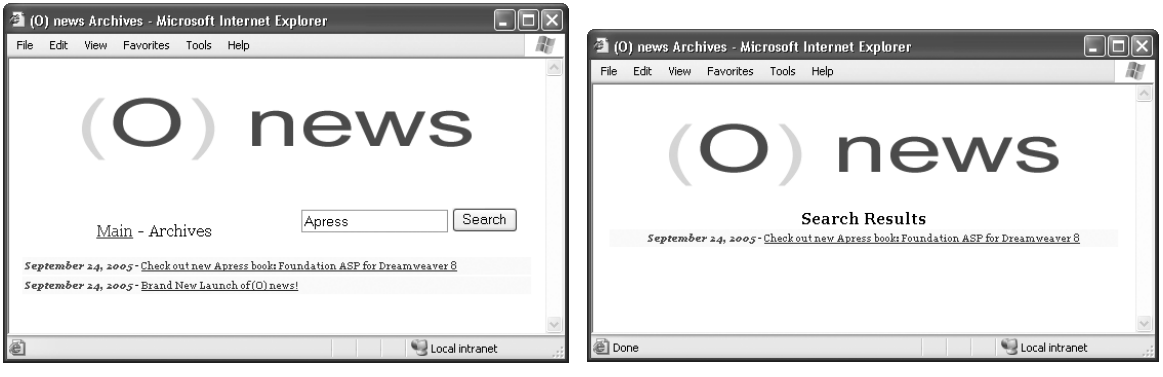
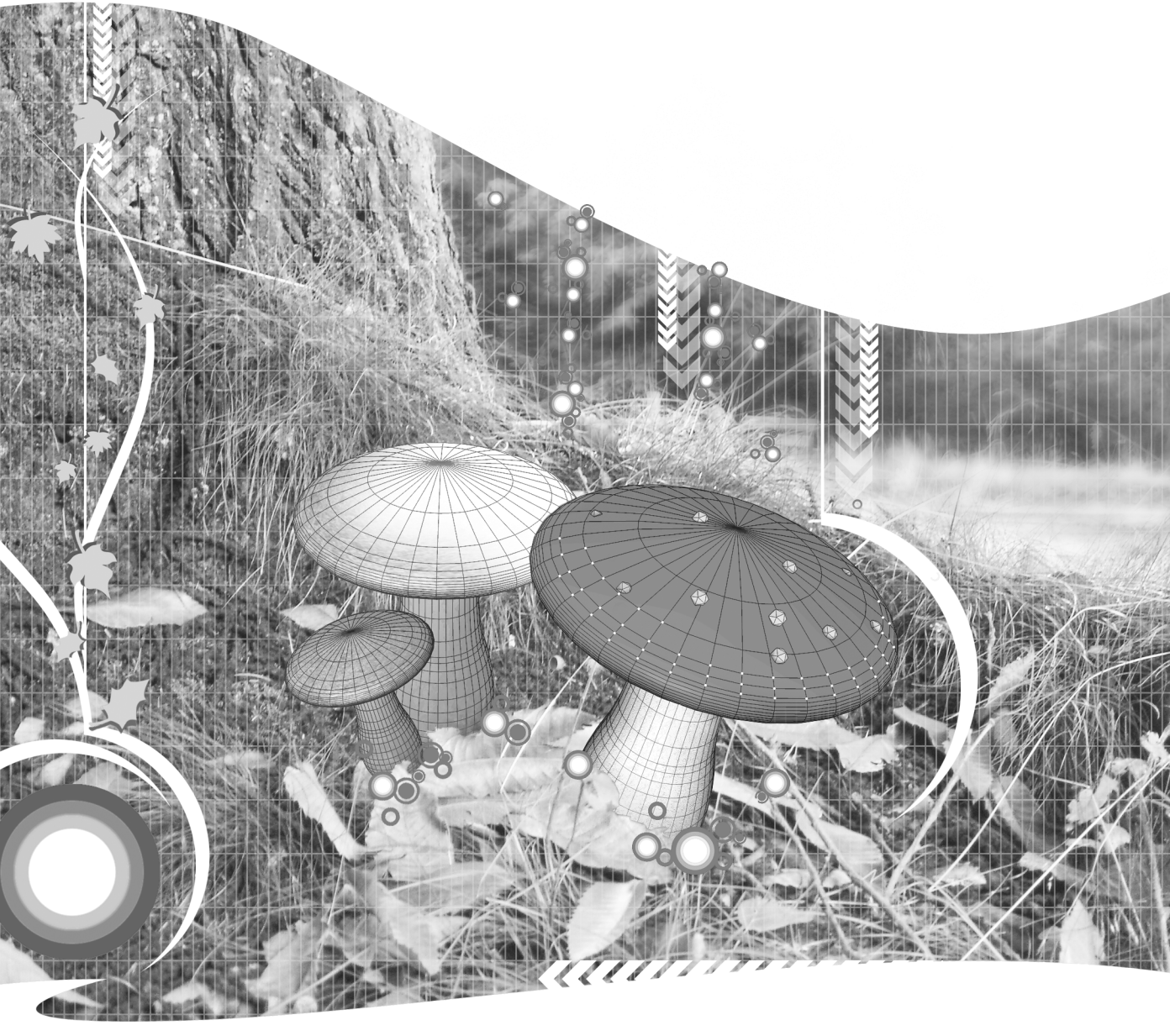

(0) news Archives - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
Main - Archives
Apress
Search
September 24, 2005 - Check out new Apress book: Foundation ASP for Dreamweaver 8
September 24, 2005 - Brand New Launch of (O) news!
Local intranet
(0) news Archives - Microsoft Internet Explorer
File Edit View Favorites Tools Help
(O) news
Search Results
September 24, 2005 - Check out new Apress book: Foundation ASP for Dreamweaver 8
Done
Local intranet

Figure 9-54. Searching for a keyword that exists in your blog entries

Conclusion

Congratulations! You have now completed your entire blog application. You've learned a great deal, including how to connect your blog database to Dreamweaver, how to display your blogs and manage your blog page, and how to secure your administration pages—you should now have the knowledge that will allow you to start tailoring your own blogs to your specific needs.

In the next chapter, you'll learn how to create a complete database-driven "photo album" web application that allows you to upload photos in your own customizable photo album folders. You'll be able to add, edit, and delete both photos and photo albums.



Chapter 10

IMAGE GALLERY

In this chapter, you'll learn how to create your own **image gallery** web application, where you can upload and store a collection of photos (usually categorized) and share them with others. You've probably seen image galleries all over the Internet; nearly all major commercial websites (e.g., Yahoo, MSN, and AOL) give their users the option to create their own personal image gallery.

Overview of the image gallery application

The image gallery you will build in this chapter will allow you to create albums, upload images into specific photo albums, and finally display those images by album. You will also be able to edit and delete photos, captions, and entire albums.

You will be using a third-party ASP component for the uploading functionality called AspUpload, which is available at `www.aspupload.com`. A free 30-day trial is available for download from the site; the full version costs $149.00 at the time of this writing. You should download and install this component before you continue the chapter (this is very easy; it just requires you to double-click an `.exe` file). Take some time to familiarize yourself with the component and its features—especially security issues—by reading the user manual at `www.aspupload.com/manual.html`.

You will be completing this web application on your local server. Whenever you decide to use this application remotely, you need to make sure the host has the AspUpload component installed on the server and available for use. Most ASP web hosting packages do come with AspUpload or some other uploading component. If your host uses something different from AspUpload, then you should familiarize yourself with that component. It will usually be just a matter of simple syntax differences, and you can make the changes accordingly.

The application is made up of four distinct areas of functionality:

- Storing information in the database
- Inserting, editing, and displaying albums
- Uploading and displaying photos
- Editing captions and deleting photos

We expand upon these topics in the sections that follow. It is a good idea to have completed Chapter 9 before completing this chapter, because you will have become more familiar with the steps that follow.

Storing information in the database

The database will not store the image files themselves; instead, it will store important information such as the name of the file and the image captions. The name of the file will then be accessed when you display the images on a web page.

Inserting, editing, and displaying albums

This area will consist of three pages. The main page will display the photo albums and links to create, and edit/delete album functionality. The second page will allow the user to create a new album in the database. The third page will allow the user to edit or delete an album.

Uploading and displaying photos

This area will consist of two pages. The first page will allow users to browse their computers for a photo and enter a related caption. This page will submit an upload form to the second page. This page will also contain links to edit or delete photos.

The second page will actually perform the upload action, in which it uploads the photo to a directory on the server and inserts the data related to the photo in the database. The second page then redirects the user to the upload page after the action is performed.

Editing captions and deleting photos

This section will consist of two pages. The first page will allow for the editing of a photo caption, and the second page will delete a photo, deleting the image file itself from the server as well as deleting any related data for that file from the database.

> *For testing purposes, this image gallery application is available at Omar's website,* `www.elbaga.net`. *Check the homepage for details.*

Creating the database for the image gallery

Now you will begin creating your image gallery. You will work through the four main areas just discussed, starting with creating the database. In our example, you will create the database in SQL Server; however, you can make simple changes to allow you to use Access instead (consult Chapter 4 for more information).

The first thing you need to do is start up SQL Server Service Manager and make sure it is running. If it is, then let's move on.

Load SQL Server Enterprise Manager, and, from the console root, create a new database and name it ophotos (or whatever you want). You could also use the same database you created the in blog application from Chapter 9, adding the new tables to it. If you do create a new database, make sure you set appropriate user permissions for the database as demonstrated in Chapter 9.

You will now create two tables for your database: `tbl_photoalbums`, which will store the names of albums, and `tbl_photos`, which will store the names of photos.

Table 1: tbl_photoalbums

To create the first table, `tbl_photoalbums`, follow these steps:

1. Right-click the name of your database from the console root and select New ➤ Table.
2. Create the following columns (also shown in Figure 10-1).

Column Name	Data Type	Length	Allow Nulls
album_id	int	4	
album_name	varchar	50	
dtstamp	datetime	8	

3. Set the `album_id` column to the primary key by right-clicking the column and selecting Set Primary Key.
4. Set the Identity value for the `album_id` column to Yes. This will ensure that the database gives each record a unique ID number.

> *For Access, change the data type for this column to* Autonumber.

5. Set the default value for the `dtstamp` column to the value of (getdate()).

> *For Access, change the data type for this column to* Date/Time *and use* `now()` *as the* Default Value.

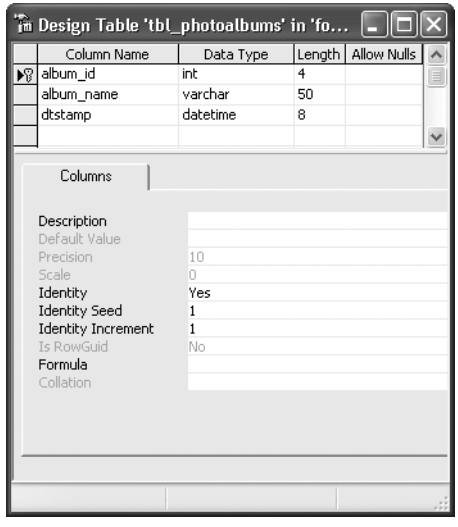


Figure 10-1. The tbl_photoalbums table in Design view in SQL Server after customization

Table 2: tbl_photos

To create the second table, `tbl_photos`, follow these steps:

1. Right-click the name of your database from the console root and select New ➤ Table.
2. Create the following columns (also shown in Figure 10-2).

Column Name	Data Type	Length	Allow Nulls
photo_id	int	4	
photo_name	varchar	100	
photo_caption	varchar	1000	X
dtstamp	datetime	8	
album_id	int	4	

3. Set the `photo_id` column to be the primary key by right-clicking the column and selecting Set Primary Key.

> *For Access, change the data type for this column to* Autonumber.

4. Set the Identity value for the `photo_id` column to the value Yes. This will ensure that the database gives each record a unique ID number.
5. Set the default value for the `dtstamp` column to the value (getdate()).

> *For Access, change the data type for this column to* Date/Time *and use* `now()` *as the* Default Value.

6. Allow a null value for the `photo_caption` column so that you can leave the photo caption blank if you decide not to enter a caption for a photo. Notice that the `album_id` will be stored in this table as a foreign key so that you can relate each photo to a particular album; you will see this in action when you develop the actual ASP pages in Dreamweaver.

> *For Access, change the appropriate data types for each database column. Change* varchar *to* Text*, and change* int *to* Number.

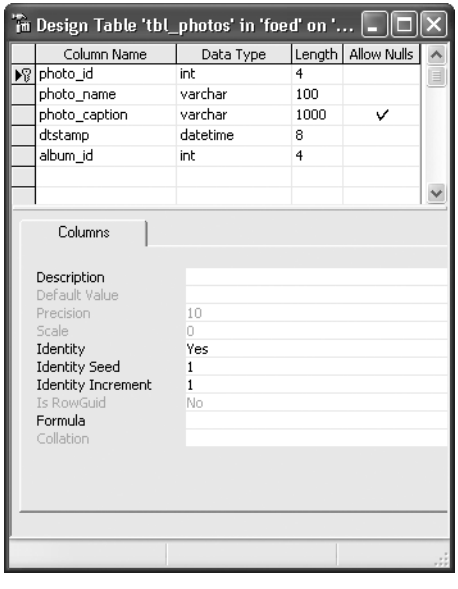


Figure 10-2. The tbl_photos table in Design view in SQL Server after customization

7. Save the table as `tbl_photos`.

The relationship between tbl_photos and tbl_photoalbums

You will now create a diagram in SQL Server that shows the relationship between the two tables.

1. Double-click your database from the console root and right-click Diagram.
2. Select New Database Diagram from the drop-down menu.
3. When the wizard appears, simply cancel out of it—you will not need to use the wizard. Instead, right-click in the empty diagram area.
4. Select Add Table and highlight the two tables you've just created, `tbl_photoalbums` and `tbl_photos`.
5. Click Add and then Close. Your diagram should now look like Figure 10-3.

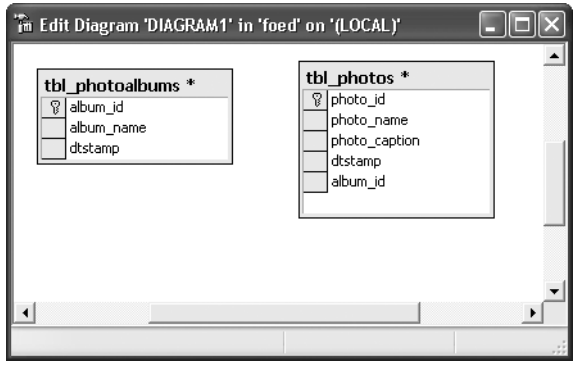


Figure 10-3. The Edit Diagram dialog box in SQL Server before columns are related

6. Highlight the primary key in the `tbl_photoalbums` table and drag it to the `tbl_photos` table.
7. You will now see the Create Relationship dialog box, which displays the relationships between the two tables (as shown in Figure 10-4). Check the box labeled Cascade Deleted Records.

   Remember that albums are stored in one table and photos are stored in another. So, when an album is deleted, related photos are not deleted. However, with this option unselected, SQL Server will automatically delete all photo entries related to an album when that photo album is deleted.

> *This relationship assumes that each image can appear in only one photo album. There are ways to modify this system to allow one image to appear in many photo albums at once, but for now we'll keep it simple and focus on a one-to-one relationship.*

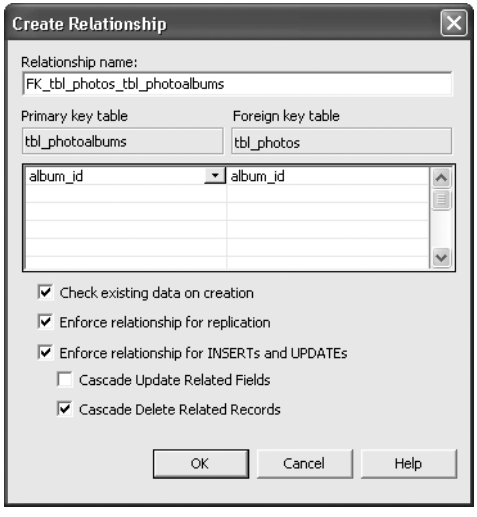


Figure 10-4. The Create Relationship dialog box in SQL Server after customization

8. When you are done, click OK. You will now see the diagram including the relationship in the diagram area, as shown in Figure 10-5.

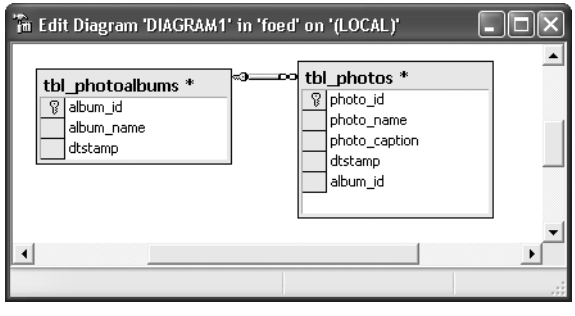


Figure 10-5. The Edit Diagram dialog box in SQL Server after columns are attached

9. Finally, save your diagram, and don't forget to set user permissions for this database if you created a new database, as discussed in Chapter 9.

Now that you've completed your database, you'll begin working in Dreamweaver. Before you start the next sections, you should define a new site in Dreamweaver or simply create a new folder named Chapter10 in the site you have previously defined in which you will view your dynamic ASP pages locally. For a refresher on how to do so, please see the section "Defining an ASP VBScript site in Dreamweaver" in Chapter 2.

Connecting Dreamweaver to your database

Before you begin creating the dynamic ASP pages, you need to create a connection to this database in Dreamweaver. Dreamweaver will then create the ASP code for a database connection and insert it in all the dynamic pages you specify.

Start Dreamweaver, create a new ASP VBScript dynamic page, and save it as `myphotos_main.asp`. Then follow these steps:

1. Select Application Panel ➤ Databases ➤ (+) ➤ Custom Connection String (Figure 10-6). You can use either an ODBC or OLE DB type connection string, or you can simply create a **Data Source Name** (DSN). In this example, you will use an ODBC connection string.

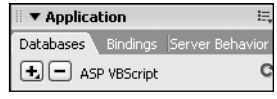


Figure 10-6. The Databases tab of the Application panel

2. Name your connection and enter the following string (be sure to replace the machine name, database name, username, and password with your own):

```
Driver={SQL Server};Server= MachineName;Database=dbname;Uid=oe;Pwd=123;
```

3. Select Using Driver on this machine.
4. Test your connection, and you should see the message Connection was made successfully (Figure 10-7).

Figure 10-7. Dreamweaver's message when a database connection has been tested and connected successfully

If you received an error, it's because Dreamweaver can't connect to the database. The cause of this problem is likely to be one of the following:

- SQL Server is not running.
- Permissions for the database are not set properly.
- The username and password entered in the string does not match the login stored in SQL Server.

If you get an error, you will need to look into it before proceeding. Review Chapter 4 for help with troubleshooting.

Connection strings

You can use either an ODBC or OLE DB connection string. Here are the strings customized for SQL Server.

ODBC:

```
Driver={SQL Server};Server= MachineName;Database=dbname;Uid=sa;
➥ Pwd=pass;
```

OLE DB:

```
Provider=SQLOLEDB;Data Source= MachineName;
➥ Initial Catalog= dbname;User ID=oe;Password=123;
```

Here are the strings, customized for Access.

ODBC:

```
Driver={Microsoft Access Driver (*.mdb)};
➥ Dbq=C:\somepath\databasename.mdb;
```

OLEDB:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
➥ C:\somepath\databasename.mdb;
```

Data Source Name (DSN)

You can create a DSN, which will link your server to the database directly without your having to use a string.

1. Choose Start ➤ Control Panel ➤ Administrative Tools ➤ Data Sources (ODBC) ➤ System DSN ➤ ADD.
2. Select the SQL Server driver.
3. Enter a name for your DSN and select your server.
4. Use SQL Server authentication and enter the username and password you created for yourself.
5. Click Next and then Finish. Test your data source and then click OK. This DSN will then be available in Dreamweaver when you create your connection.

Note that if you decide to use a DSN, you will need to have your web host create a DSN for you when you upload your local pages to your remote server, which may or may not be convenient. Also, most web hosts allow a limited number of DSNs to be created, which will limit your database connections if you want to add more databases to your website in the future. When you use a connection string, you do not need to create a DSN since you are manually creating the connection string, but you will need the SQL Server name and the username and password for your database on the server.

Now that you have a successful connection between Dreamweaver and the database, you can proceed to create your dynamic pages.

Inserting, editing, and displaying albums in the image gallery

In this section, you will begin creating the Dreamweaver pages to create new photo albums, edit/delete photo albums, and display existing photo albums to your web users. Let's get started with building your first page.

Creating myphotos_main.asp

This page will retrieve the albums from your `tbl_photoalbums` database table. It will display the titles for the albums, and each album will be hyperlinked to a second page that will display the photos contained in that album, which you will learn more about in the next section.

1. In Dreamweaver, open up the `myphotos_main.asp` page you previously created.
2. Select Application Panel ➤ Server Behaviors ➤ Recordset (Query) to open the Recordset dialog (Figure 10-8).
3. Stay in Simple mode, and add the following information into the dialog:
   - **Name**: rsgetphotoalbums
   - **Connection**: The connection to the appropriate database
   - **Table**: dbo.tbl_photoalbums
   - **Columns**: All
   - **Filter**: None
   - **Sort**: dtstamp in Descending order

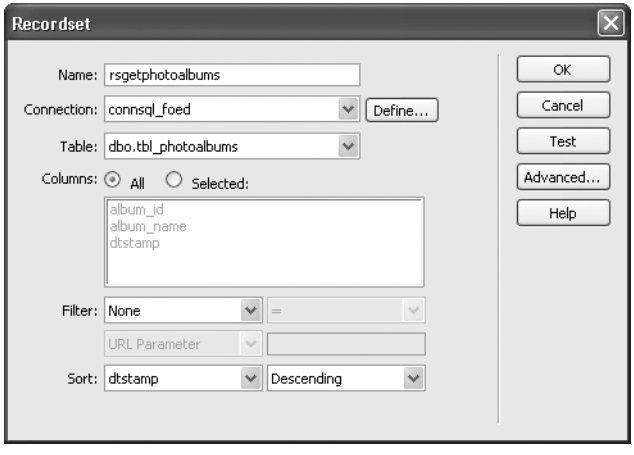


Figure 10-8. The rsgetphotoalbums recordset in Dreamweaver after customization

4. Test the recordset and you should see the results in a new Dreamweaver dialog box. You will probably see no results because you did not populate the database with any data yet.
5. Click OK to save the recordset.

Once you've created your recordset, the rest of the page is merely a matter of aesthetics. Once this page loads, the albums will be retrieved from the database. Now you just need to design how the page is going to display the records, which we'll cover in the next section.

> *Notice that the names of SQL Server database tables need to be prefixed with* dbo. *Here is an example of how a SQL database table will be referred to within a SQL command:* dbo.tbl_photoalbums *and* dbo.tbl_photos. *However, in Access,* dbo *is not needed. In a SQL command connecting to an Access database, those tables will be referred to as simply* tbl_photoalbums *and* tbl_photos.

Designing the main web page

Follow these steps to design the primary web page:

1. In Dreamweaver, switch to Design view, and insert two tables stacked over each other. The first table should have two rows and one column; the second table should have one row and one column. Set the tables to a length of 500 pixels.
2. In between the two tables, insert a horizontal rule or any sort of identifiable separation, such as a straight line.

> *In subsequent sections, text in all caps represents placeholders that will be replaced with dynamic text.*

3. Insert the text Photo Albums Create New! in the first row of the first table.
4. Insert the following text in the second row of the first table:

   Welcome to your online photo album!

   Clicking the link labeled "New" above will allow you to create new albums.
5. Insert the text ALBUM-NAME -edit in the single cell of the second table. Because this text will use dynamic hyperlinks, you will link this text at the end of this section. Your page should now look something like Figure 10-9.

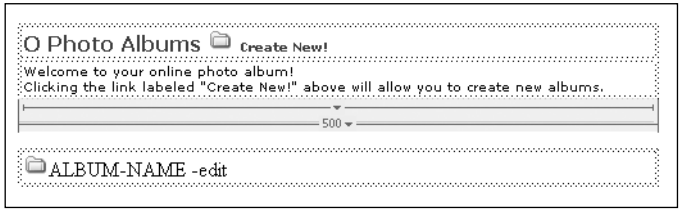

O Photo Albums
Create New!
Welcome to your online photo album!
Clicking the link labeled "Create New!" above will allow you to create new albums.
500
ALBUM-NAME -edit

Figure 10-9. The myphotos_main.asp page in Dreamweaver before dynamic content is added

Implementing the dynamic features of the main web page

Follow these steps to implement the dynamic features of the web page:

1. Highlight the text ALBUM-NAME and then go to Application Panel ➤ Bindings.
2. Expand the recordset named `rsgetphotoalbums` and select the album_name column, as shown in Figure 10-10.
3. Click Insert at the bottom of the Application panel. The dynamic album name from the recordset now replaces the previous ALBUM-NAME static text.

Figure 10-10. Accessing dynamic data in Dreamweaver. Here you retrieve the value of the album_name column.

Before you complete this page, you need to link some of the text to wire up the links to their appropriate pages.

4. Highlight the Create New! text from your web page and link it to `addalbum.asp`, which you will create later in this section. This will take the user to a page that allows him or her to add a photo album.
5. Now highlight the edit text. You need to link this text to `edit_photo_album.asp`, which will allow you to edit albums; you will create this later on as well. Remember that in order to edit a record, you need to find it—you can find a single record by its unique ID. The unique ID number for each album has been retrieved by the recordset `rsgetphotoalbums`. You need to link to the edit page and pass this ID as a variable in the URL querystring.
6. From the Properties panel, click the yellow folder to create a hyperlink. Link it to `edit_photo_album.asp`.
7. Click the Parameters button and pass a variable named `aid` with the value being retrieved from the `album_id` column in the recordset (Figures 10-11 and 10-12). This will pass the album ID in the URL querystring to the page that will do the editing.

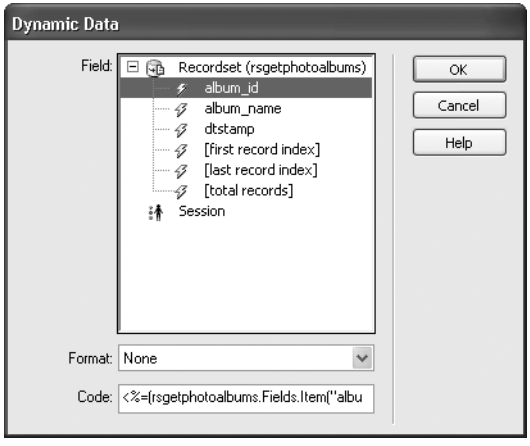


Figure 10-11. Accessing dynamic data in Dreamweaver. Here you retrieve the value of the album_id column.

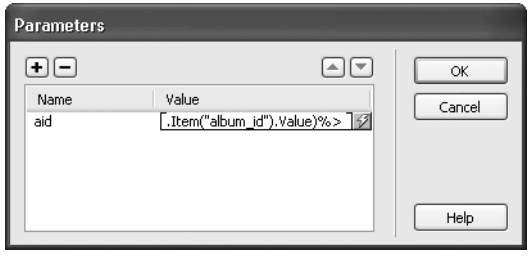


Figure 10-12. The Parameters dialog box in Dreamweaver after a URL parameter has been added

Now you need to link the album name to a page that will display the photos.

8. In Design view and in the document, highlight the dynamic text placeholder for the `album_name` column and link it to `upload.asp`, which you will create later in this section (see Figure 10-15).

9. You must again pass the album ID as a URL parameter in this link, so that the receiving page displays photos for the relative album. Add a URL parameter named `aid` and select the dynamic album ID from the `rsgetphotoalbums` recordset as the value.

   Before you close this page, you have a few more steps to complete. You need apply a repeat region for the album name so that all photo albums are listed and you will only show the table if records are actually returned in the recordset. If there are none, then no records will display.

10. Highlight the row of the second table that includes the dynamic album name and go to Applications ➤ Server Behaviors ➤ (+) ➤ Repeat Region.
11. When the dialog box appears, select the rsgetphotoalbums recordset, select All records, and click OK (Figure 10-13).

Figure 10-13. Setting the Repeat Region dialog in Dreamweaver. Here you specify all records to show.

12. Next, highlight the entire second table and go to Applications ➤ Server Behaviors ➤ (+) ➤ Show Region ➤ Show Region If Recordset Is Not Empty.
13. When the dialog box appears, select the rsgetphotoalbums recordset and click OK (Figure 10-14).

Figure 10-14. Selecting the appropriate recordset in the Show Region dialog

You have now completed the main page of the O Photos Albums image gallery (Figure 10-15). Here, the user will be able to view albums and go to any section needed to control the photos.

If you want to display a message when no records are found, you can simply add a custom message such as No albums have been created yet anywhere in the document (but not within the region that shows when records *are* found) and complete step 12, but use the Show Region If Recordset Is Empty behavior instead.

Figure 10-15. The image gallery page in Dreamweaver after all dynamic behaviors have been added

When you test the page in your browser, you should see something like Figure 10-16.

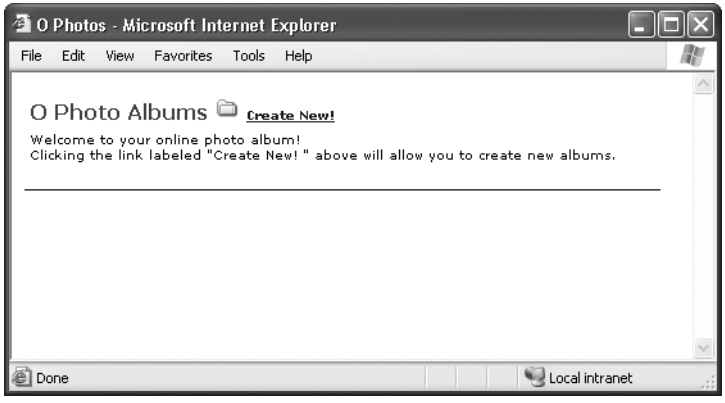

O Photos - Microsoft Internet Explorer
File Edit View Favorites Tools Help
O Photo Albums Create New!
Welcome to your online photo album!
Clicking the link labeled "Create New! " above will allow you to create new albums.
Done
Local intranet

Figure 10-16. The myphotos_main.asp page in a web browser

No albums appear because you haven't yet added any. You haven't even created the dynamic web page to allow for the addition of albums, but you will do that next. If you were to open the database itself through Enterprise Manager and populate the table with albums, they would appear here.

Creating addalbum.asp

This page will give you the ability to actually add albums to display. When an album is added, it will then display on the `myphotos_main.asp` page.

In Dreamweaver, create a new dynamic ASP VBScript page and save it as `addalbum.asp`. You will not need to add a recordset to this page because the behavior you will use when you create the dynamic side will do all the work. Next, you'll build the HTML side of the page.

Designing the Add Album page

All you need in this page is a form with one text field and a form button.

1. In Dreamweaver, switch to Design view and insert one table with four rows and one column.
2. In the first row, enter the text O Photos - Add a new album.
3. Link the O Photos text back to the `myphotos_main.asp` page.
4. In the third row, enter the following text: Provide a name for your new album or leave the default setting. Clicking the button below will create your new album.
5. In the fourth row, insert a form and name it frm_insert. Within the form, type the text Add Album; directly below it, insert a text field named txtphotoalbumname.
6. Enter the following code as the default value of this text field:

```
<%=Date()%>
```

7. This will automatically provide a default value of today's date for the album name in case the user does not enter a specific album name.

8. Below this text field, insert a form button and set its value to Add Album. You are now finished with the HTML side of the web page, and it should look like Figure 10-17.

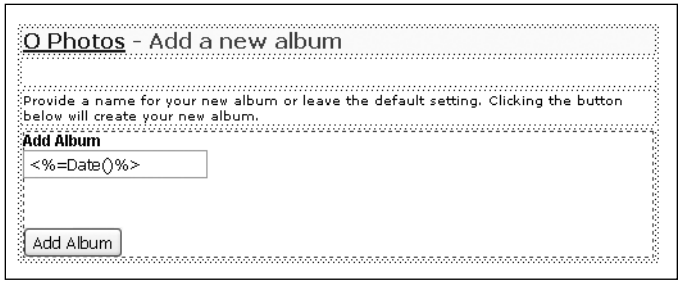

O Photos - Add a new album
Provide a name for your new album or leave the default setting. Clicking the button below will create your new album.
Add Album
<%=Date()%>
Add Album

Figure 10-17. The addalbum.asp page before the dynamic aspect has been implemented

Wiring up the Add Album page

You'll now add the page functionality to insert the form contents into the database when it's submitted.

1. Go to Application Panel ➤ Server Behaviors ➤ (+) ➤ Insert Record to open the Insert Record dialog box (Figure 10-18).

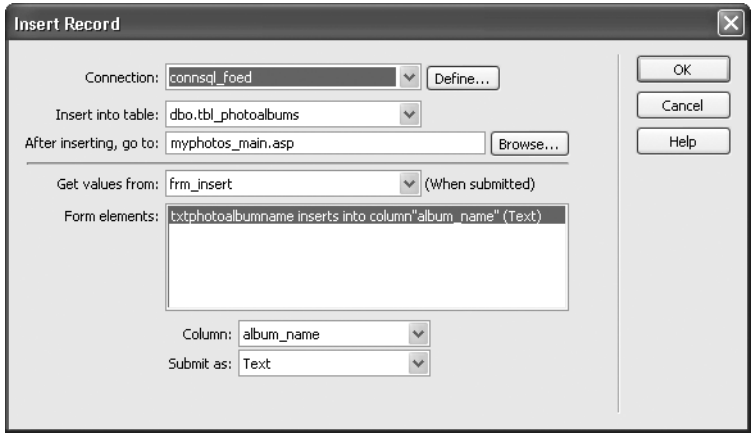


Figure 10-18. The Insert Record dialog after its customization

2. Fill in this dialog box as follows, and then click OK:
   - **Connection**: The connection to your database
   - **Insert into table**: dbo.tbl_photoalbums
   - **After inserting, go to**: myphotos_main.asp
   - **Get values from**: frm_insert
   - **Form elements**: The text field labeled `txtphotoalbumname` should insert into the column name `album_name`

- **Column**: album_name
- **Submit as**: Text

You have now completed your addalbum.asp page. When you access this page, you will be able to add albums to the database. Test this out in your web browser. Remember to surf to the myphotos_main.asp page first and then click the Create New! link, which will take you to addalbum.asp, as shown in Figure 10-19.

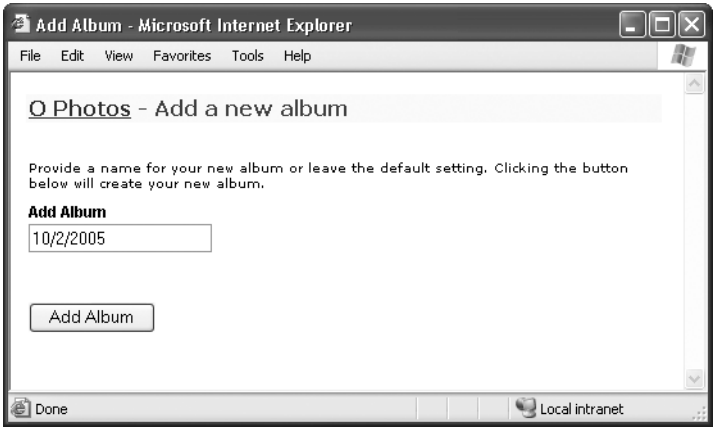

Add Album - Microsoft Internet Explorer
File Edit View Favorites Tools Help
O Photos - Add a new album
Provide a name for your new album or leave the default setting. Clicking the button below will create your new album.
Add Album
10/2/2005
Add Album
Done
Local intranet

Figure 10-19. The addalbum.asp page after the dynamic aspect has been created

Go ahead and insert a couple of albums to test it out (Figure 10-20).

O Photos - Microsoft Internet Explorer
File Edit View Favorites Tools Help
O Photo Albums Create New!
Welcome to your online photo album!
Clicking the link labeled "Create New! " above will allow you to create new albums.
Canada Trip 2005 -edit
Hesham's Graduation -edit
Local intranet

Figure 10-20. The myphotos_main.asp page after new albums have been added

Bear in mind that the page that allows for the viewing or editing/deleting of an album has yet to be created, so the edit links on myphotos_main.asp will currently result in a 404 error. You'll create the editing/deleting page next.

Creating edit_photo_album.asp

This page will retrieve an album selected from the `myphotos_main.asp` page and allow you to edit or delete it.

First, in Dreamweaver, create a new dynamic ASP VBScript page and save it as `edit_photo_album.asp`. You will now create a recordset to retrieve the appropriate record. Remember that you need to retrieve the record whose album ID equals the variable passed in the URL querystring named `aid`.

1. Go to Application Panel ➤ Server Behaviors ➤ (+) ➤ Recordset (Query) to open the Recordset dialog box (Figure 10-21).

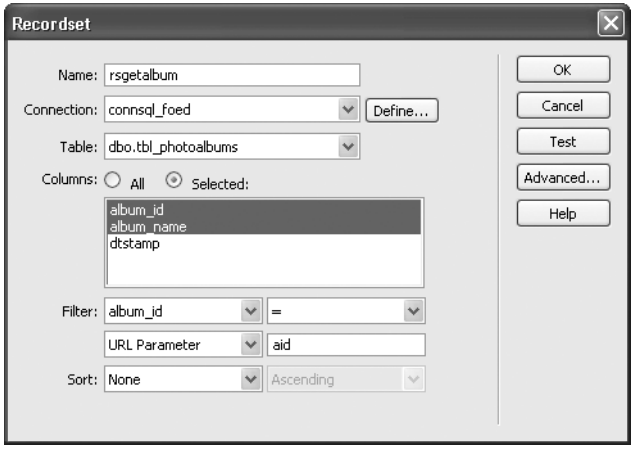


Figure 10-21. The Recordset dialog after customization

2. Fill in the dialog box as follows, and then click OK:
   - **Name**: rsgetalbum
   - **Connection**: The connection to your database
   - **Table**: dbo.tbl_photoalbums
   - **Columns**: album_id and album_name
   - **Filter**: Filter where the `album_id` is equivalent to a URL parameter named `aid`
   - **Sort**: None

Your recordset is now complete.

Implementing the dynamic features

Let's go straight to implementing the dynamic features of this page; this should take care of most of the HTML as well.

1. Enter a header for the page—O Photos, or whatever you like. Link this text back to the `myphotos_main.asp` page.
2. Go to Insert ➤ Application Objects ➤ Update Record ➤ Record Update Form Wizard to open the Record Update Form dialog box (Figure 10-22).

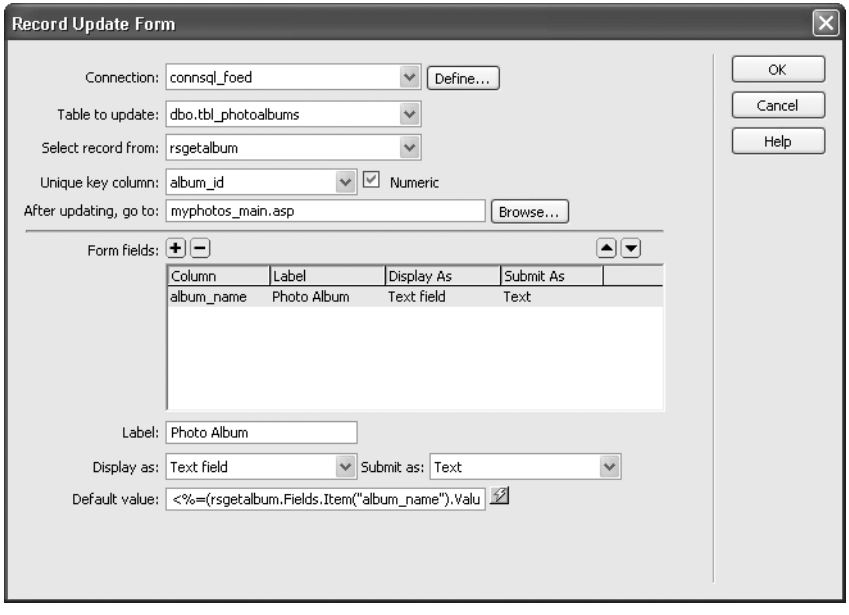


Figure 10-22. The Record Update dialog after its customization

3. Fill in the dialog box as follows, and then click OK:
   - **Connection**: The connection to your database
   - **Table to update**: dbo.tbl_photoalbums
   - **Select record from**: rsgetalbum
   - **Unique key column**: album_id
   - **After updating, go to**: myphotos_main.asp
   - **Form fields**: album_name
   - **Label**: Photo Album
   - **Display as**: Leave default value
   - **Submit as**: Leave default value
   - **Default value**: Leave default value

   Dreamweaver will create the HTML form and add the Update Record behavior in one fell swoop. Now you need to give the user the ability to delete the album.

4. In Design view, place your cursor beneath the update table.
5. Insert a table with one row and one column.
6. Insert a form into the table labeled `frm_delete`.
7. Within the form place a form button with a label of Delete Album.
8. Now you'll add the Delete Record server behavior to this form. Select Application Panel ➤ Server Behaviors ➤ Delete Record to open the Delete Record dialog (Figure 10-23).

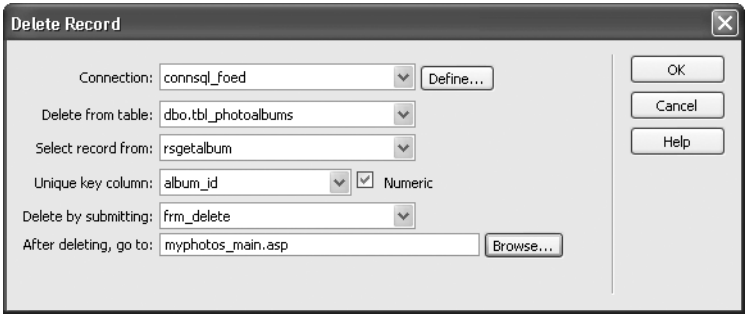


Figure 10-23. The Delete Record dialog after settings have been changed

9. Fill in the dialog box as follows, and then click OK:
   - **Connection**: The connection to your database
   - **Delete from table**: dbo.tbl_photoalbums
   - **Select record from**: rsgetalbum
   - **Unique key column**: album_id (keep Numeric selected)
   - **Delete by submitting**: frm_delete
   - **After deleting, go to**: myphotos_main.asp

Your `edit_photo_album.asp` page is now complete and should look like Figure 10-24 in Design view. You have now finished the second main component to your image gallery application. Test this out now in your browser. You will now be able to insert, edit, and delete albums.

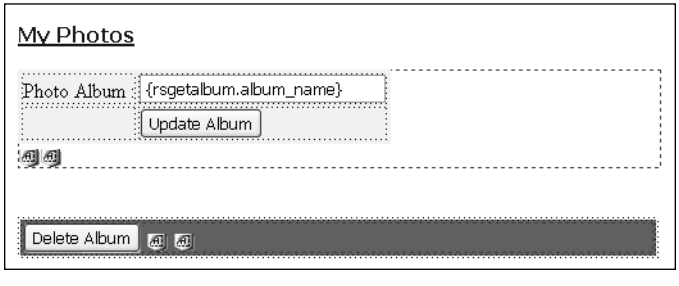


Figure 10-24. The edit_photo_album.asp page in Dreamweaver after the dynamic features have been implemented

Creating pages for uploading and displaying photos

The pages you are about to create will give you the ability to upload and display photos within albums. The first page, `upload.asp`, will allow a user to browse for an image file on his or her local computer and enter a caption for the photo. The form will then be submitted to a second page, `upload_action.asp`, which will upload the image to a specified directory on the server and insert the photo details in the `tbl_photos` database table. Bear in mind that in this example, the photos will also be displayed on the `upload.asp` page.

Before you start creating the pages, create a new folder within the root of your site labeled uploads. This is where uploaded images will be stored.

Creating upload.asp

Pay close attention in this section, because this is one of the most important components within the application. This page will have two functions: it will first submit image files and corresponding captions to the `upload_action.asp` page, and then it will display the photos related to the particular album being viewed.

In Dreamweaver, create a new dynamic ASP VBScript page and save it as `upload.asp`. You need to first retrieve the photos for the selected album. Remember that the `album_id` is in the querystring that was passed through the album name hyperlink on `myphotos_main.asp`.

1. Select Application Panel ➤ Server Behaviors ➤ (+) ➤ Recordset (Query) to open up the Recordset dialog box (Figure 10-25).

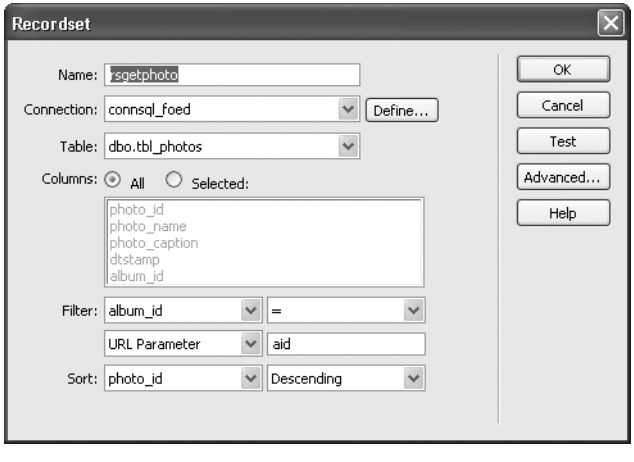


Figure 10-25. The Recordset dialog after customization

2. Stay in Simple mode and fill in the dialog box as follows, before clicking OK:
   - **Name**: rsgetphoto
   - **Connection**: The connection to the appropriate database
   - **Table**: dbo.tbl_photos (or tbl_photos for Access)
   - **Columns**: All
   - **Filter**: Filter the `album_id` whose value is equivalent to a URL parameter named `aid`
   - **Sort**: photo_id in Descending order

Test the recordset and you should see the results in a new Dreamweaver dialog box. This recordset will now retrieve all photos related to the `album_id` passed in the querystring. You will probably see no results because you have probably not yet populated the database with any data.

Designing upload.asp

Follow these steps to design the `upload.asp` page:

1. In Dreamweaver, switch to Design view and create a header, this time with the text O Photos > Upload Photo.
2. Link the O Photos text back to the `myphotos_main.asp` page so the user has an easy way to get back to the main page.
3. Below the O Photos image or your own custom logo, insert a form named `frm_upload`. Set Action to upload_action.asp, Method to POST, and Enctype to multipart/form-data (Figure 10-26).

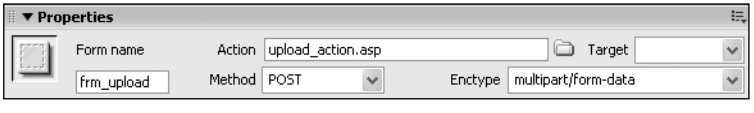


Figure 10-26. The Properties panel while the frm_upload form is selected

4. Place your cursor inside the form and insert a table with one row and two columns.
5. With your cursor in the left column, insert a file field named file and a text area named txt_caption.
6. Place your cursor in the right column and insert a form button with a label of Upload Photo. Next to the form button, insert a hidden field named hdn_albumid.
7. Add a second table with one row and one column below the form. Place your cursor in the bottom table and enter the text IMAGE.
8. Insert a line break and then insert the text PHOTO-CAPTION edit caption – delete.

You are now finished with the HTML portion of your page. It should look like Figure 10-27.

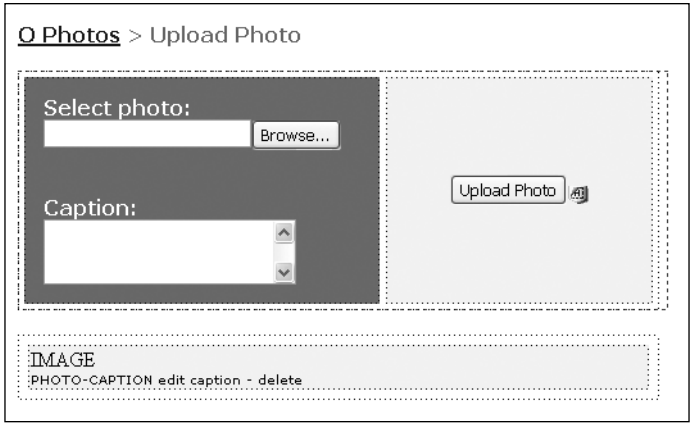


Figure 10-27. The upload.asp page in Dreamweaver before the dynamic features have been implemented

Wiring up upload.asp

Now you're ready to wire up the `upload.asp` page.

1. Highlight the `hdn_albumid` hidden field and insert the following code as its value:

   ```
   <%=Request.QueryString("aid")%>
   ```

   This will populate the hidden field with the album ID residing in the querystring. This will later be inserted into the `tbl_photos` table when the photo is uploaded, to relate the photo to the album selected.

2. In Design view, highlight the IMAGE text and then go to Insert ➤ Image Objects ➤ Image Placeholder.

3. Click OK when the Image Placeholder pops up, and delete the IMAGE text.

4. Highlight the Image Placeholder, switch to Split view, and insert the following code as the value for the `src` attribute of the `<img>` HTML tag:

   ```
   uploads/<%=(rsgetphoto.Fields.Item("photo_name").Value)%>
   ```

   This is where the image URL will come from—photos will be uploaded to a folder named uploads, which you should create now within the root of this web application, and the `photo_name` column will contain the name of the photo, including its file extension. The uploads folder will be empty until you actually start uploading photos there.

5. Delete the `name`, `width`, and `height` attributes from the `<img>` tag, and give the image a border of 2 using the Properties panel.

6. Highlight the PHOTO-CAPTION text and go to Application Panel ➤ Bindings.

7. Expand the `rsgetphoto` recordset and highlight the photo_caption column (Figure 10-28).

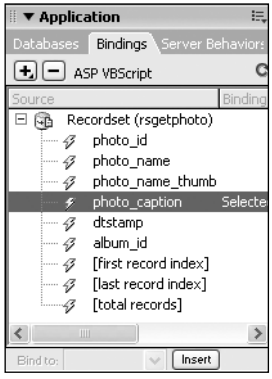


Figure 10-28. The Bindings tab in the Application panel while photo_caption is highlighted

8. Click the button labeled Insert at the bottom of the panel. This will replace the text with the dynamic photo caption.
9. Highlight the edit caption text and link it to `edit_caption.asp` (which you will create in the next section), making sure you set it to pass a URL parameter named `pid` with its value equal to the `photo_id` column of the `rsgetphoto` recordset (Figure 10-29). The link to the edit caption text should look like this:

```
edit_caption.asp?pid=<%=(rsgetphoto.Fields.Item("photo_id").Value)%>
```

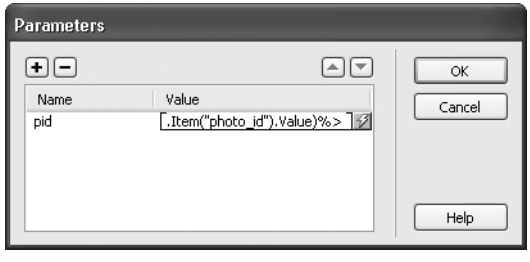


Figure 10-29. The Parameters dialog after adding the name and value of the URL parameter pid

10. Now you need to highlight the delete text and link it to the `delete_action.asp` page, which you will create in the next section.
11. While you're linking, make sure you use the Parameters button to set it up to pass three URL parameters in the querystring (Figure 10-30).

Figure 10-30. The Parameters dialog after adding the name and value of all URL parameters

Here are the three parameters and their values:

Name	Value
pid	<%=(rsgetphoto.Fields.Item("photo_id").Value)%>
pn	<%=(rsgetphoto.Fields.Item("photo_name").Value)%>
aid	<%=(rsgetphoto.Fields.Item("album_id").Value)%>

What you are doing here is passing the current photo ID, photo name, and album ID to `delete_action.asp`. This page needs these values in order to delete the correct photo and return to the correct photo album after a photo has been uploaded. You will create this page in the next section, but make sure you create the hyperlink on this page correctly. The final link to the `delete` text should look like this:

```
delete_action.asp?pid=<%=(rsgetphoto.Fields.Item("photo_id").Value)%>&
➥ pn=<%=(rsgetphoto.Fields.Item("photo_name").Value)%>&
➥ aid=<%=(rsgetphoto.Fields.Item("album_id").Value)%>
```

The final job to do here is add Repeat Region and Show Region behaviors, so the photos display correctly.

12. Highlight the row of the bottom table that includes the image and add the Repeat Region behavior, selecting to display all records from the `rsgetphoto` recordset.
13. Highlight the entire bottom table and add a Show Region If Recordset Is Not Empty behavior.

You are now finished with the upload.asp page. It should look like Figure 10-31 in Design view and Figure 10-32 when you test it in a web browser.

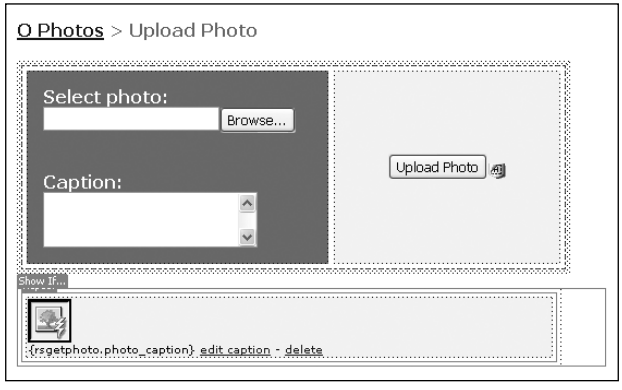


Figure 10-31. The upload.asp page in Dreamweaver after the dynamic side has been implemented

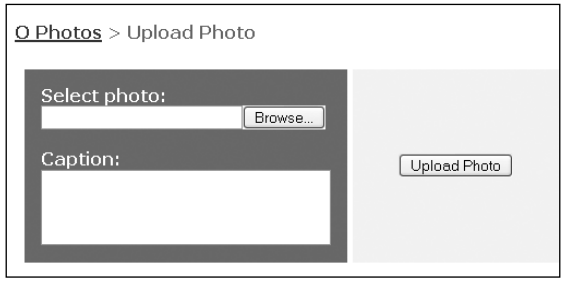


Figure 10-32. The upload.asp page in a web browser after the dynamic side has been implemented

You can now go to your web browser and click an album on your myphotos_main.asp page—this will take you to your upload.asp page. Bear in mind that no photos will appear, and you won't be able to upload any yet because you haven't completed the upload_action.asp page. You'll create this next.

Creating upload_action.asp

This page will actually allow you to upload photos. It will have two functions: uploading the selected file from upload.asp to the appropriate folder on the server and inserting the photo details in the tbl_photos table.

This page will then redirect the user back to upload.asp, which will then display the photos. This page will contain no HTML, because it will only execute the appropriate action and then redirect.

Create a new dynamic ASP VBScript page and save it as upload_action.asp. You will create a command that will automatically insert the photo details passed in the querystring into the tbl_photos table without form submission on the part of the user. You will insert three values into the table—the photo name, photo caption, and album ID—which all are passed to upload_action.asp from the upload.asp form.

1. Switch to Code view and select Application Panel ➤ Server Behaviors ➤ (+) ➤ Command, which will open the Command dialog (Figure 10-33).

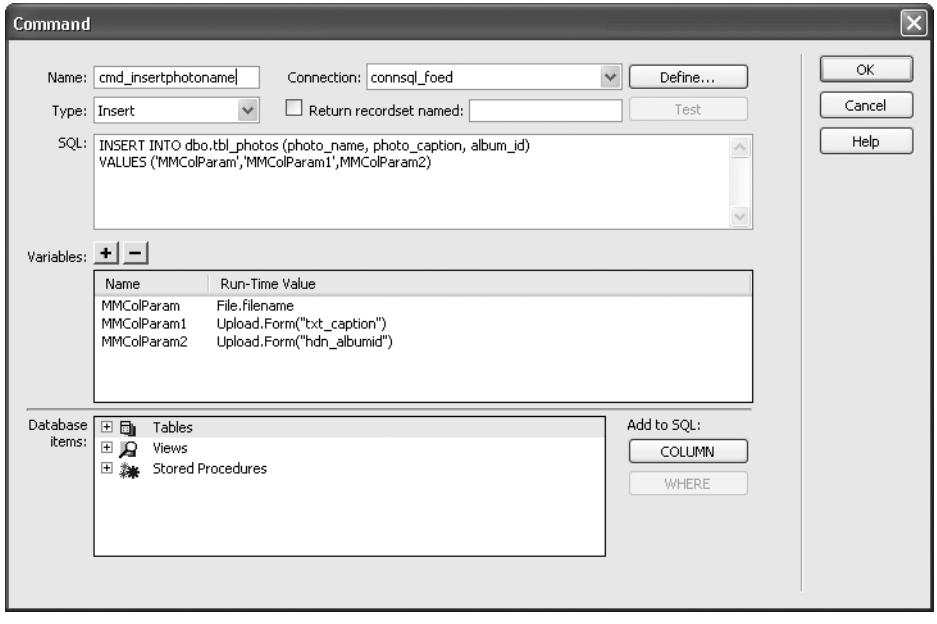


Figure 10-33. The Command dialog box

2. Fill in this dialog box as follows, and then click OK:
   - **Name**: cmd_insertphotoname
   - **Connection**: The connection to your database
   - **Type**: Insert
   - **SQL**:

```
INSERT INTO dbo.tbl_photos (photo_name, photo_caption, album_id)
VALUES ('MMColParam','MMColParam1',MMColParam2)
```

   - **Variables**:

Name	Run-Time Value
MMColParam	File.filename
MMColParam1	Upload.Form("txt_caption")
MMColParam2	Upload.Form("hdn_albumid")

The runtime values are syntax specific to the AspUpload component. Here you are collecting the photo caption, file name, and album ID submitted through the form on the previous page and inserting them into the appropriate database columns.

Now you need to tweak the code slightly to perform the actual upload and redirect the page. First, let's make sure the upload occurs before the command.

3. Switch to Code view and add the following code directly **above** the Command behavior:

```
<%
  Set Upload = Server.CreateObject("Persits.Upload")
  'Do not overwrite files with same name,
  'create unique name automatically
  Upload.OverwriteFiles = False
  'Save file
  Upload.Save "<ENTER PHYSICALPATH TO SITE ROOT>\uploads\"
  'Create File Object to attain file properties
  Set File = Upload.Files("file")
%>
```

Figure 10-34 should help you find this location more easily. This code contains syntax specific to the AspUpload component. You need to specify the physical path to the directory on the server where the uploading will occur. When you use this web application on a remote server, you can attain the physical path of the directory from your web host. While this application is being tested locally, the physical path should look something like C:\Inetpub\wwwroot\website\uploads\.

You can easily find out the physical path to a web page by inserting the following code in any ASP page. The physical path on the server to that page will then be printed to the screen.

```
<% Response.Write Server.MapPath("filename.asp") %>
```

Replace `filename.asp` *with the name of the page that you want to find out the path to.*

```
1  <%@LANGUAGE="VBSCRIPT" CODEPAGE="1252"%>
2  <!--#include file="../Connections/connsql_foed.asp" -->
3
4  <%
5  Set Upload = Server.CreateObject("Persits.Upload")
6  'Do not overwrite files with same name, create unique name automatically
7  Upload.OverwriteFiles = False
8  'Save file
9  Upload.Save "C:\Inetpub\wwwroot\foed\c10-upload\uploads\"
10 'Create File Object to attain file properties
11 Set File = Upload.Files("file")
12 %>
13 <%
14
15 if(File.filename <> "") then cmd_insertphotoname__MMColParam = File.filename
```

Figure 10-34. AspUpload code in Dreamweaver's Code view in upload_action.asp

Finally, you need to redirect the user after these actions have occurred.

4. In Design view, scroll to a line directly **after** the Command behavior, as shown in Figure 10-35, and enter the following code:

```
<%
  Response.Redirect("upload.asp?upload=success&
➥  fn=" & File.filename & "&aid=" & Upload.Form("hdn_albumid"))
%>
```

```
30  cmd_insertphotoname.Execute()
31  %>
32  <%
33  'Redirect to upload.asp passing appropriate Parameters
34  Response.Redirect("upload.asp?message=success&fn=" & File.filename &
    "&aid=" & Upload.Form("hdn_albumid"))
35  %>
```

Figure 10-35. The redirect code in Dreamweaver's Code view in upload_action.asp

This code redirects the user back to the `upload.asp` page, making sure to pass the appropriate parameters (file name and album ID) in the URL querystring, so the correct photos and photo album are retrieved when the user is actually taken back to the `upload.asp` page, and a confirming message is displayed.

Now you can view the page live in your web browser. Start out by surfing to the `myphotos_main.asp` page. Select an album and then try to upload a few images. In each case, you should be taken back to the `upload.asp` page with the correct photos displaying. This process should look something like that shown in Figures 10-36 and 10-37.

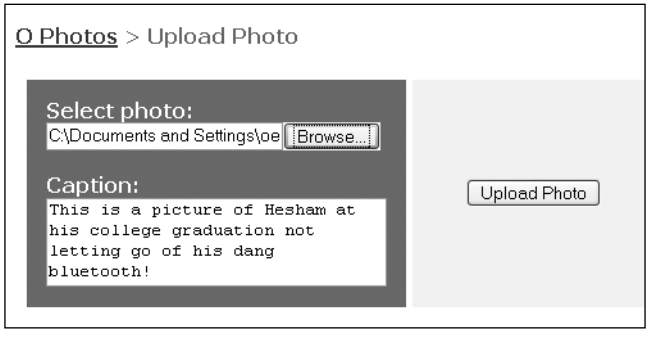

O Photos > Upload Photo
Select photo:
C:\Documents and Settings\pe
Browse...
Caption:
This is a picture of Hesham at his college graduation not letting go of his dang bluetooth!
Upload Photo

Figure 10-36. The upload.asp page in a web browser with the upload form filled out

Upload - Microsoft Internet Explorer
File Edit View Favorites Tools Help
O Photos > Upload Photo
Select photo:
Browse...
Caption:
Upload Photo
This is a picture of Hesham at his college graduation not letting go of his dang bluetooth! edit caption - delete
Done
Local intranet

Figure 10-37. The upload.asp page in a web browser after a photo has been uploaded

Keep in mind that you will still not be able to edit the captions or delete the photos because you haven't created the appropriate pages for this yet. However, after inserting a photo, hover your cursor over the edit caption and delete links on the `upload.asp` page, and make sure you see the correct parameters in the URL in the status bar of the web browser.

Now you just need to create the pages to allow for editing captions and deleting photos.

Creating the Edit Caption and Delete Photo pages

Now you're in the home stretch—the last major part of the application you need to create is the pages that allow you to edit photo captions and delete photos.

Creating edit_caption.asp

The `edit_caption.asp` page will allow you to update the captions inserted when the photos were originally uploaded.

Create a new dynamic ASP VBScript page and save it as `edit_caption.asp`. You now need to create a recordset to retrieve the appropriate record containing the caption you want to edit. Remember that the edit caption link on `upload.asp` passes the photo ID as a URL parameter in the querystring, which will give you the ability to filter the database for the appropriate record.

1. Go to Application Panel ➤ Server Behaviors ➤ (+) ➤ Recordset (Query) to open the Recordset dialog box (Figure 10-38).

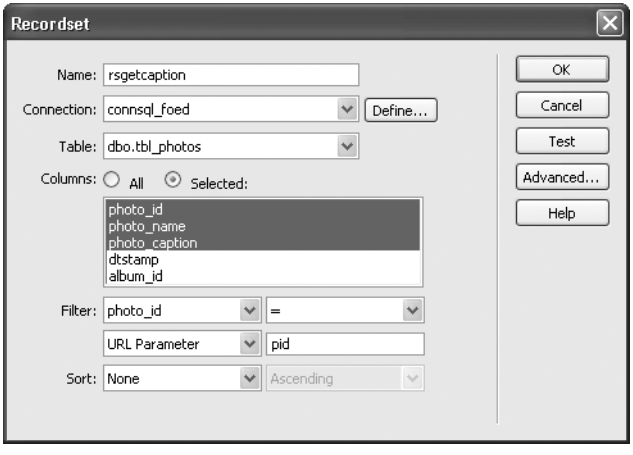


Figure 10-38. The Recordset dialog after customization

2. Stay in Simple mode, fill out the dialog box as follows, and then click OK:
   - **Name**: rsgetcaption
   - **Connection**: The connection to the appropriate database
   - **Table**: dbo.tbl_photos
   - **Columns**: photo_id, photo_name, photo_caption
   - **Filter**: Filter the `photo_id` whose value is equivalent to a URL parameter named `pid`
   - **Sort**: None

   This recordset will now retrieve the photo whose caption you wish to edit. The next step is to add an update record form.

3. Select Insert ➤ Application Objects ➤ Update Record ➤ Record Update Form Wizard to open the Record Update Form dialog (Figure 10-39).

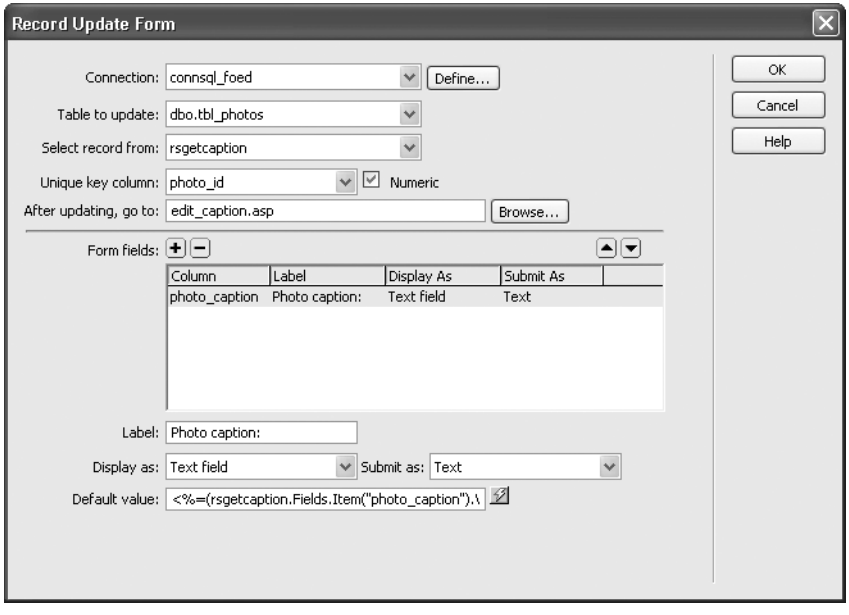


Figure 10-39. The Record Update Form dialog after customization

4. Fill in this dialog box as follows, and then click OK:
   - **Connection**: The connection to your database
   - **Table to update**: dbo.tbl_photos
   - **Select record from**: rsgetcaption
   - **Unique key column**: photo_id
   - **After updating, go to**: edit_caption.asp
   - **Form fields**: Photo_caption
   - **Label**: Photo caption:
   - **Display as**: Leave default value
   - **Submit as**: Leave default value
   - **Default value**: Leave default value

Dreamweaver will create the HTML form and add the Update Record behavior in one fell swoop. If you want, even though the user will know the photo he or she is attempting to update, you can also go to Bindings and drag and drop the name of the photo on this page as an extra useful piece of functionality, as shown in Figure 10-40.

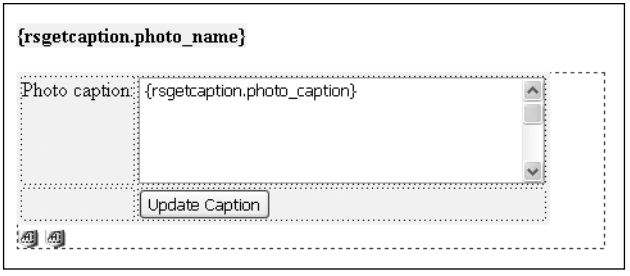


Figure 10-40. The edit_caption.asp page in Dreamweaver after the dynamic side has been implemented

Now try testing the page in your web browser by clicking the edit caption link below a selected photo. You will currently be able to edit the caption but not delete the photo (see Figures 10-41 and 10-42), because the page that allows for deleting has not been created yet.

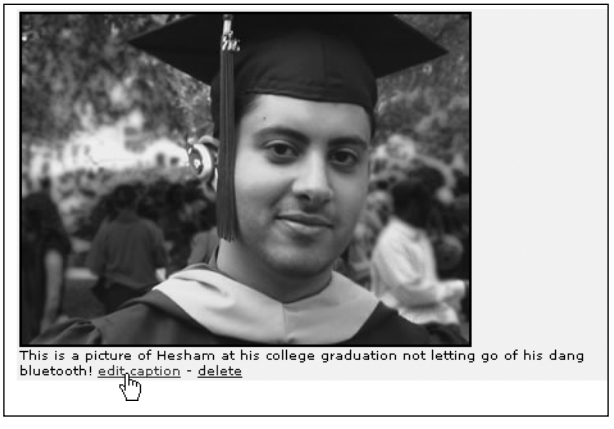


Figure 10-41. The upload.asp page while selecting the edit caption link of a particular photo

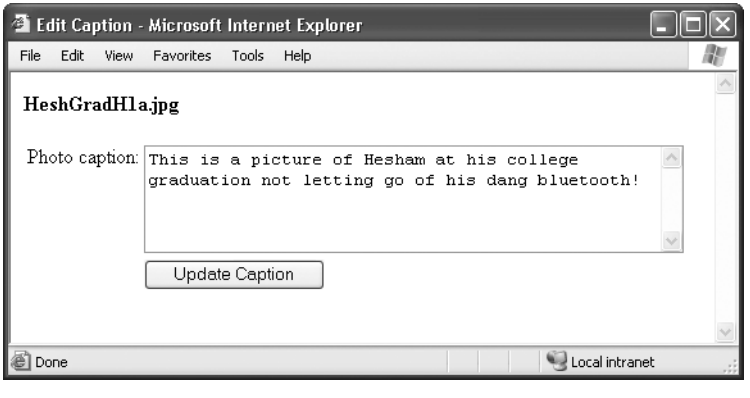

Edit Caption - Microsoft Internet Explorer
File Edit View Favorites Tools Help
HeshGradH1a.jpg
Photo caption:
This is a picture of Hesham at his college graduation not letting go of his dang bluetooth!
Update Caption
Done
Local intranet

Figure 10-42. The edit_caption.asp page when a particular record has been selected

You have now completed `edit_caption.asp`. The only page left to create is the page to delete photos, `delete_action.asp`.

Creating delete_action.asp

This page is similar to the `upload_action.asp` page in that it will contain no HTML—just some AspUpload syntax to delete the photo from the server and a command to delete the photo details from the database and redirect back to the appropriate page.

You will create a command that will automatically delete the photo details passed in the querystring from the `tbl_photos` table without any form submission needed on the part of the user. The entire record will be deleted from the table, and the user will then be redirected back to the `upload.asp` page, along with the URL parameters, so that `upload.asp` can find the correct album to display.

Create a new dynamic ASP VBScript page, save it as `delete_action.asp`, and follow these steps:

1. Switch to Code view and select Application Panel ➤ Server Behaviors ➤ (+) ➤ Command to open the Command dialog (Figure 10-43).

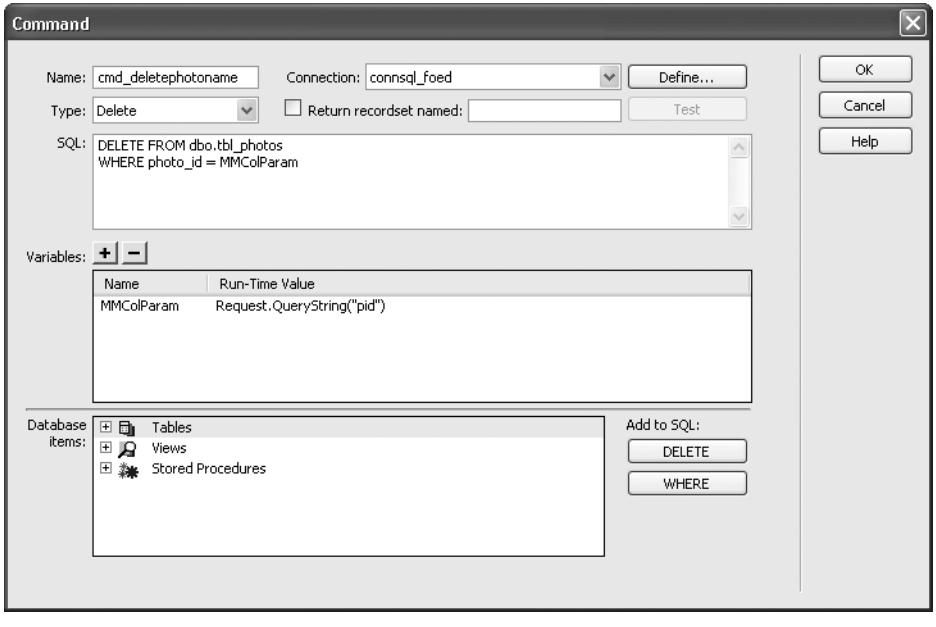


Figure 10-43. The Command dialog after customization

2. Fill out the dialog box as follows, and then click OK:
   - **Name**: cmd_deletephotoname
   - **Connection**: The connection to your database
   - **Type**: Delete

- **SQL**:

```
DELETE FROM dbo.tbl_photos
WHERE photo_id = MMColParam
```

- **Variables**:

Name	Run-Time Value
MMColParam	Request.QueryString("pid")

Here you are deleting the record containing the photo ID passed in the querystring when the delete link is selected on the previous page. Remember that the delete link should be passing three URL parameters to this page: the photo ID, photo name, and album ID.

Before executing this page, make sure that these three parameters are actually being passed via the delete *link on* upload.asp*! You need to pass the photo ID because with it you are telling the database which record you want it to retrieve. Without it, the database won't know which record you want. You need to pass the photo name because you will use that to delete the appropriate photo from the* uploads *folder on the server. Lastly, you need to pass the album ID so that when you redirect back to the* upload.asp *page, you redirect the user back to the appropriate photo album. Remember that the* upload.asp *page is retrieving photos for a specific album, so without passing the album ID again, the page will not be able to take the user back to the correct photo album.*

Now you need to tweak the code slightly to perform the actual delete and redirect the page. First, let's make sure the delete occurs before the command.

3. Switch to Code view and add the following code directly **above** the Command behavior (refer to Figure 10-44 for a better idea of where to position it):

```
<%
  Set Upload = Server.CreateObject("Persits.Upload")
  'Delete File
  Upload.DeleteFile "< PHYSICAL PATH OF DIRECTORY HOLDING UPLOADED
➥  IMAGES>" & Request.QueryString("pn")
%>
```

```
3
4   <%
5   Set Upload = Server.CreateObject("Persits.Upload")
6   'Delete File
7   Upload.DeleteFile "C:\Inetpub\wwwroot\foed\cl0-upload\uploads\" &
    Request.QueryString("pn")
8   %>
9   <%
10
11  if(Request.QueryString("pid") <> "") then
```

Figure 10-44. The AspUpload code in Dreamweaver's Code view in delete_action.asp

This code contains syntax specific to the AspUpload component. You concatenated the photo name passed in the querystring and the physical path to find the appropriate file on the server. When you use this web application on a remote server, you can attain the physical path of the directory from your web host. While this application is being tested locally, the physical path should look something like this: C:\Inetpub\wwwroot\website\uploads\.

Finally, you need to redirect the user back to upload.asp after these actions have occurred.

4. Scroll to the line directly **after** the Command behavior and enter the following code:

```
<%
  'Redirect back to upload.asp passing appropriate URL Parameters
  Response.Redirect("upload.asp?message=deleted&fn=" &
➥  Request.QueryString("pn") & "&aid=" & Request.QueryString("aid"))
%>
```

Figure 10-45 should clarify where to put the code.

```
22  cmd_deletephotoname.Execute()
23  %>
24  <%
25  'Redirect back to upload.asp passing appropriate URL Parameters
26  Response.Redirect("upload.asp?message=deleted&fn=" & Request.
    QueryString("pn") & "&aid=" & Request.QueryString("aid"))
27  %>
```

Figure 10-45. The redirect code in Dreamweaver's Code view in delete_action.asp

This redirects the user back to the upload.asp page, making sure to pass the appropriate parameters (file name and album ID) in the URL querystring so the correct photos are retrieved for display on upload.asp when the redirect is completed and a message confirming the deletion is shown.

Save the page, and then test your finished web application. Start out by surfing to the myphotos_main.asp page. Select an album and then try to upload a few images. You should be taken back to the upload.asp page with the correct photos displayed. Now you can edit captions and delete photos if you so desire. The file will be deleted from the server, and the photo details will be deleted from the database.

Conclusion

Congratulations! You've completed your entire image gallery application, which gives you the ability to insert, edit, delete, and display individual photos and entire photo albums!

ASP web applications can range from very simple to very complex. The purpose of this book has been to get you up and running with dynamic websites. Remember that the front-end design and structure of your web application can be customized to your liking using CSS as well as custom HTML structures.

> *For a great guide on CSS and XHTML web design using Dreamweaver 8, check out* Foundation Web Design with Dreamweaver 8 *by Craig Grannell (friends of ED, 2006).*

There are endless choices as to how you could design the front-end. For example, in this chapter's photo gallery example, the form to upload photos is on the same page that displays photos. You could have the upload form on a separate page. You could even have photos for a particular album appear on the page that displays the names of photo albums so that when the album is selected, the related photos appear underneath the folder.

When you become more proficient at designing these web applications, you will be able to easily integrate your own front-end design into the web application. Never forget, however, that it is usability that makes web applications successful in the real world. All your efforts can go to waste if people don't know how to use the web application. For example, if users can't find the form to upload photos, they won't be uploading any and will soon leave your site in search of a more intuitive solution.

In addition, as you become more proficient with the concepts of a database and how to design it to successfully deliver content to web pages, you will be able to constantly add new features. For example, you could easily allow users to add comments to individual photos or albums if you tweak the database and apply the concepts you learned in this book.

Reading this book has given you a great foundation on which to build your skills. To continue your learning process, we recommend that you submerge yourself in Dreamweaver by reading the Dreamweaver documentation that comes along with the application, and checking out the latest news about Dreamweaver 8 on `www.macromedia.com`. Also become a frequent visitor of community sites such as DMXzone.com (`www.dmxzone.com`) and Community MX (`www.communitymx.com`), both of which offer loads of tutorials, articles, and extensions by dedicated writers. Much of the content is free, while some premium content is available for a fee.

For ASP resources, you should visit the following sites:

- 4GuysFromRolla.com (`www.4guysfromrolla.com`): Includes thousands of articles on ASP development
- Aspin (`www.aspin.com`): Includes thousands of turnkey ASP components and scripts
- DevGuru (`www.devguru.com`) and W3Schools (`www.w3schools.com`): Good resources for online ASP and VBScript references

Join mailing lists or newsgroups such as the official Macromedia Dreamweaver newsgroup, which you can find at `www.macromedia.com/support/forums` under Product Forums. You can interact with the forum online in your web browser or through a newsreader such as Outlook Express or Thunderbird. There are hundreds of regular users with similar interests, goals, and questions as you, and access to Dreamweaver experts and their knowledge is easily at hand. Just make sure you follow good etiquette practices in the forums:

- Don't post the same question too many times and to different groups simultaneously.
- Make sure the subject of your post is clear and gives adequate information about exactly what you want to know. "Quick question" or "How do you . . ." are subject lines that are easily ignored because they don't tell the reader about your actual query.
- Keep your post clear and concise, making it easy for others to read and figure out how your query can be addressed.

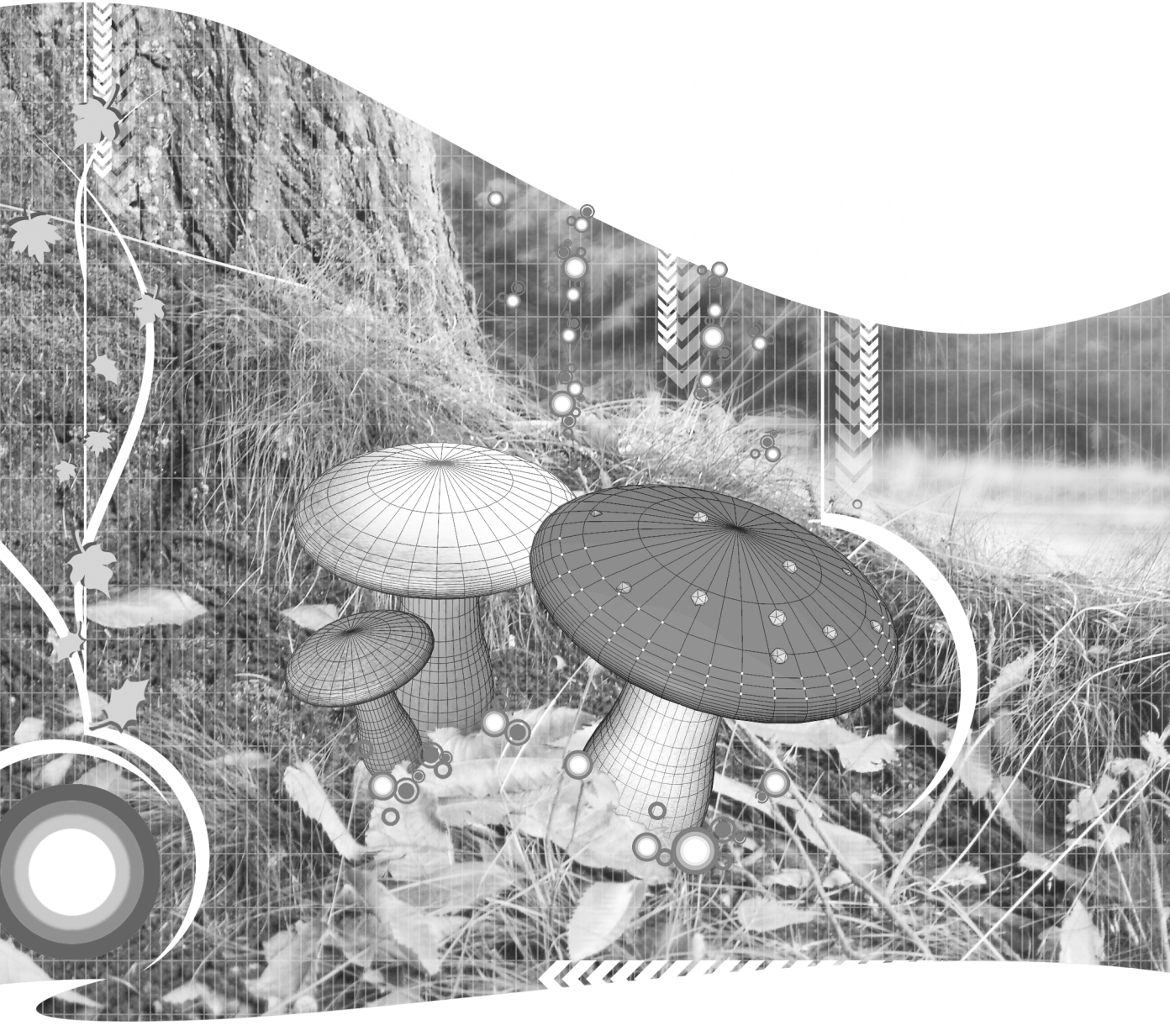
Be sure to frequent the Macromedia Exchange for Dreamweaver at www.macromedia.com/cfusion/exchange. Here you can find downloadable extensions that add new features to Dreamweaver's current capabilities.

Also, read more books. We'd like to recommend the following:

- *Foundation Web Design with Dreamweaver 8* by Craig Grannell (friends of ED, 2006)
- *Foundation PHP for Dreamweaver 8* by David Powers (friends of ED, 2005)

Remember that a Google search is always at hand.

Last but not least, you can contact the authors of this book if you have any queries, comments, or problems concerning this book. Omar Elbaga is available through his site at www.elbaga.net. Rob Turnbull is available through his site at www.robgt.com.



INDEX

Numbers and symbols

! (exclamation point)
  code problems, 200
<% %> ASP delimiters, 46

A

Abandon statement, ASP Session
  deleting session variables, 79
Access key field
  Input Tag Accessibility Attributes dialog, 204, 205
access levels
  defining, 229
  Dreamweaver recording, 231
Access, Microsoft
  creating Access database, 84
  creating relationships in, 98–100
  creating tables, 93–95
  database connection with Dreamweaver, 115
  handling dates, 113
  naming conventions, 104
  queries, 102
  setting up DSN to, 114
  SQL Server compared, 87
accessibility features
  Input Tag Accessibility Attributes dialog, 178, 204, 205
  turning off, 178
Action attribute, form tag, 134
Add separator button, Insert bar, 33
Add user form, 234
AddAddress method, AspEmail
  sending cc and bcc, 154
  sending e-mail, 152
AddAddress property, AspEmail, 152
addalbum.asp, 301–303
AddCc method, AspEmail, 154
adding_numbers.asp, 55
addition operator, VBScript, 55, 65
AddRecipient property, JMail
  sending cc and bcc, 155
  sending e-mail, 155
AddRecipientBcc property, JMail, 156
AddRecipientCc method, JMail, 155
AddReplyTo property, AspEmail, 154
adduser.asp
  registering new users, 233
administration
  blogs, 266–276
  creating login system, 225–232
  restricting access to pages, 223–243
administrators database table, 224
ADO cursor types, 217
ADO recordset information, 217
Advanced Recordset builder dialog, 121
After inserting, go to field
  Insert Record dialog box, 179
albums
  see image gallery web application
All view, CSS panel, 21
allquotes.asp page
  adding Log Out User server behavior, 231
  linking to delete.asp, 190
  linking to editquote.asp, 184, 188
  login system restricting access, 229
  registering new users, 234
  Repeat Region server behavior, 182
AND keyword, SQL, 105
AND operator, VBScript, 62
andor.asp, 62, 63
anonymous domains
  general requirements for mail components, 147
Application panel, 16, 22–25
  Bindings panel, 16, 23
    blog security, 279
    context menu illustrated, 209
    myphotos_main.asp, 298
    onews_main.asp, 257
    pasting recordset using panel group, 209
    upload.asp, 310
  Components panel, 16, 25
  Databases panel, 16, 22
    connecting Dreamweaver to blog database, 253
    connecting Dreamweaver to image gallery database, 294
  Server Behaviors panel, 16, 24

Application tab, Insert bar, 14–16
Command icon, 14
Delete Record icon, 15
Dynamic Data icon, 14
Go To icon, 15
Insert Recordset icon, 15
Master Detail Page Set icon, 15
Recordset icon, 14
Recordset navigation status icon, 15
Recordset Paging icon, 15
Repeated Region icon, 14
Show as Menu view, 11
Show Region icon, 15
Update Record icon, 15
User Authentication icon, 16
XSL Transformation icon, 16
array functions, VBScript, 58
ASC keyword, SQL
ORDER BY clause, 110
ASP
ASP brackets, 5
commenting code, 57
cookies, 72–76
creating simple dynamic ASP page, 4
description of basic function, 5
Dreamweaver 8 benefits for developer, 9
environment variables, 79–80
features supported by Dreamweaver, 10
resources, 323
scripting ASP in JavaScript, 46
server models supported by Dreamweaver, 8
session variables, 76–79
specifying scripting language, 46
ASP blocks/delimiters <% %>, 46, 47
ASP directive
Dreamweaver inserting, 46
specifying scripting language, 46
ASP JavaScript server model
getting support for, 8
ASP mail components, 146
ASP shortcut (<%=)
outputting VBScript variables, 50
ASP tab, Insert bar, 12–14
Code Block icon, 13
If, Else If, Else, End icons, 13
Include icon, 12
Output icon, 13
Response.Write icon, 13
Server Variables icon, 12
Server.CreateObject icon, 13
Show as Tabs view, 12
Trimmed Form Element icon, 13
Trimmed QueryString Element icon, 13
ASP variables
see variables, VBScript
ASP VBScript
see under VBScript
AspEmail
formatting e-mail with HTML in, 153
properties and methods
AddAddress method, 152
AddAddress property, 152, 154
AddCc method, 153
AddReplyTo property, 154
FromName property, 154
Host property, 152
sending cc and bcc with, 153
sending e-mail with, 151–154
AspUpload code
delete_action.asp, 321
downloading, 287
upload_action.asp, 314
Assets panel, 28
assignment operators, VBScript, 61–62
attributes
form tag, 134
Attributes panel, 26
authentication
User Authentication icon, 16
authentication methods
configuring SMTP server, 144
Author drop-down menu
binding dynamic select lists with selected item, 211
building, 206
building dynamic select lists, 204
AuthorID field, tblQuotes
foreign keys, 196
authors
adding to edit page, 210
AuthorID field, tblQuotes, 176
creating insertauthor.asp, 199
Authors recordset
building, 203
updating editquote.asp page, 208
Authors table
see tblAuthors table
Autonumber data type
creating primary keys in Access, 94

B

behaviors, 27
see also server behaviors
Behaviors panel, 26
BETWEEN keyword, SQL, 108

Bindings panel, 16, 23
  blog security, 279
  building Quotes recordset, 217, 218
  building SQL statements with Recordset builder, 116, 119
  context menu illustrated, 209
  myphotos_main.asp, 298
  onews_main.asp, 257
  pasting recordset using panel group, 209
  "remember me" feature, 240
  upload.asp, 310
blind carbon copies (bcc)
  sending with AspEmail, 153
  sending with CDO, 151
  sending with JMail, 155
blog security
  timeouts, 280
blogs, 245–285
  administration, 266–276
    onews_admin_archives.asp, 266–271
    onews_admin_details.asp, 270–276
    overview, 247
  application overview, 246–247
  database
    connecting Dreamweaver to, 253–254
    creating, 247, 253
    creating database tables, 249–250
    overview, 246
    populating database tables, 250–251
    setting user permissions for tables, 251
    tbl_onews table, 249, 251
    tbl_users table, 250, 251
  description, 246
  displaying blog entries, 255–266
    onews_archives.asp, 264–266
    onews_details.asp, 261–263
    onews_main.asp, 255–260
    overview, 246
  login/logout functionality, 276–280
    overview, 247
  O News blog application, 246
  searching blogs, 281–284
    overview, 247
  security, 276–280
  third-party solutions, 245
Boolean data type, VBScript, 53
Browser Check button
  Document toolbar, 34
browser view
  addalbum.asp, 303
  edit_caption.asp, 319
  myphotos_main.asp, 301
  onews_admin_archives.asp, 270, 280
  onews_admin_details.asp, 275
  onews_archives.asp, 266, 284
  onews_details.asp, 263
  onews_main.asp, 260
  onews_searchresults.asp, 284
  upload.asp, 312
  upload_action.asp, 315, 316
bugs
  avoiding Dreamweaver bug, 177
  get method, 134
buttons
  sending e-mail via form button, 158–160
byte data type, VBScript, 53

C

carbon copies (cc)
  sending with AspEmail, 153
  sending with CDO, 151
  sending with JMail, 155
cascading
  creating relationships in SQL Server, 101
case sensitivity
  programming languages, 48
  VBScript, 47, 48
categories
  adding to edit page, 210
  CategoryID field, tblQuotes, 176
  creating insertcategory.asp, 201
Categories recordset
  building, 204
  updating editquote.asp page, 208
Categories table
  see tblCategories table
Category drop-down menu
  binding dynamic select lists with selected item, 212
  building, 207
CategoryID field, tblQuotes
  foreign keys, 196
CDO (Collaboration Data Objects)
  CDONTS and CDOSYS, 147
  formatting e-mail with HTML in CDO, 150
  properties and methods
    From property, 149
    myMail.Bcc property, 151
    myMail.Cc property, 151
    Send method, 149
    Subject property, 149
    TextBody property, 149
    To property, 149
  sending cc and bcc with CDO, 151
  sending e-mail with, 148–151

CDOSYS, 147
char data type
  creating tables in SQL Server, 96
check boxes
  creating form elements with post method, 139–141
Check In icon, Files panel, 27
Check New Username dialog, 235
Check New Username server behavior, 235
Check Out icon, Files panel, 27
Code Block icon
  ASP tab, Insert bar, 13
Code view, 10
  Document toolbar and, 34
Coding toolbar, 36–37
Collaboration Data Objects
  see CDO
columns
  database naming conventions, 92
Command dialog
  delete_action.asp, 320
  upload_action.asp, 313
Command icon
  Application tab, Insert bar, 14
  inserting/updating/deleting records, 15
commands, SQL
  DELETE, 128–130
  INSERT, 124–125
  manipulating database records, 122–130
  UPDATE, 125–128
commenting code
  ASP, 57
  VBScript, 57
comparison operators, VBScript, 64–65
  description, 61
Components panel, 16, 25
concatenation, VBScript, 54, 55
concurrency
  SQL Server/Access compared, 87
conditional code
  Insert Record server behavior, 180
conditional comments
  after insertion, 188
  after insertion of quote, 181
conditional logic
  menu element, 138
conditional logic, VBScript, 68–71
  Else keyword, 69, 71
  ElseIf keyword, 69
  If...Then...Else statement, 69
  If...Then...ElseIf statement, 69
  If statement, 68
  If, Else If, Else, End icons, 13
Configuration field
  sending e-mail with CDO, 149
connections
  see database connections
ContentType property
  formatting e-mail with HTML in JMail, 155
context menu illustrated, 209
conversion functions, VBScript, 58–59
convert_to_int.asp, 59
cookies, ASP, 72–76
  deleting, 76
  expiration of cookies, 75
  Netscape's legal notice about, 76
  "remember me" login feature, 241
  Request.Cookies statement, 73
  Response.Cookies statement, 73
  updating, 75
  warning about using, 76
COUNT keyword, SQL, 107
CREATE DATABASE command
  creating SQL Server database, 85–88
Create Relationship dialog
  creating relationships in SQL Server, 101
  image gallery web application, 293
CreateObject icon
  ASP tab, Insert bar, 13
CSS panel, 21
currency data type, VBScript, 53
Current view, CSS panel, 22
cursor types, 217
Customize Favorite Objects dialog, 33

D

data
  Dynamic Data dialog box, 184
  Dynamic Data icon, 14
data checking
  Dreamweaver 8 benefits for ASP developer, 9
data exchange
  dynamic websites, 2
data types, VBScript, 52
  specifying, 54
  web browser view of, 54
database connections
  connecting with Dreamweaver, 115
    to blog database, 253–254
    to image gallery database, 294–295
  creating, 113–116
  Databases panel, 22
  Dreamweaver 8 benefits for ASP developer, 9
  setting up DSN to Access database, 114
  setting up DSN to SQL Server database, 114
Database Diagram
  relationships in SQL Server, 100

Database Properties dialog
creating SQL Server database, 87
SQL Server database, 248
database records
adding values of rows, 107
building SQL statements with Recordset builder, 116–122
choosing rows in a range, 108
choosing rows with set values, 110
choosing subset of rows, 108
counting number of rows affected, 107
deleting from database, 128–130
grouping rows into categories, 111
ignoring duplicated rows, 112
inserting into database, 124–125
manipulating, 122–130
ordering rows, 109
reversing statement logic when choosing, 112
selecting, 103–106
updating in database, 125–128
databases
Access queries, 102
adding data to, 177
creating Access database, 84
creating blog database, 247–253
creating for image gallery, 289–293
creating relationships in Access, 98–100
creating relationships in SQL Server, 100–102
creating SQL Server database, 85–88
creating tables, 92–97
designing, 91–97
handling dates, 113
introduction, 83
naming conventions, 91
relational databases, 97–103
SQL Server views, 102
SQL Server, preparing for Internet use, 88–91
Databases panel, 16, 22
connecting Dreamweaver to blog database, 253
connecting Dreamweaver to image gallery database, 294
date data type, VBScript, 53
date functions, VBScript, 58
dates, SQL, 113
Delete Record dialog
edit_photo_album.asp, 306
illustrated, 190
onews_admin_details.asp, 274
Delete Record icon
Application tab, Insert bar, 15
Delete Record server behavior, 189–192
DELETE statement, SQL, 128–130
Delete web page illustrated, 192
delete.asp page
Delete Record server behavior, 189
linking from allquotes.asp, 190
delete_action.asp, 320–322
delete_cookie.asp, 76
delimiters
ASP delimiters <% %>, 46
DESC keyword, SQL
ORDER BY clause, 110
Design view
addalbum.asp, 301, 303
Document toolbar and, 34
Dreamweaver page in, 10
edit_caption.asp, 319
edit_photo_album.asp, 306
Guides feature, 37
Hand tool, 37
login.asp, 277
Magnification list, 37
myphotos_main.asp, 297
onews_admin_archives.asp, 268, 269
onews_admin_details.asp, 272, 275
onews_archives.asp, 265, 281
onews_details.asp, 262, 263
onews_main.asp, 256, 260
Refresh Design View button, 34
searching blogs, 281
tbl_photoalbums, 290
tbl_photos, 291
tools, 37
upload.asp, 309, 312
Zoom tool, 37
Dim keyword, VBScript, 49
Dimac Development
sending e-mail with JMail, 154
Disconnect (from remote host) icon
Files panel, 27
display_sex.asp, 138
DISTINCT keyword, SQL, 112
divide operator, VBScript, 65
divide_operators.asp, 66
Do...Loop statement, VBScript, 71
Document toolbar, 33–35
Browser Check button, 34
button functionality, 34
File Management button, 34
Live Data View button, 34
Preview/Debug in Browser button, 34
Refresh Design View button, 34
Validate Markup button, 34
View Options button, 34
Visual Aids button, 35
double data type, VBScript, 53
dowhile.asp, 72

Dreamweaver
  ASP features supported by, 10
  community sites, 323
  creating new document, 3
  creating database connection, 115
  defining ASP VBScript site in, 39–42
  Macromedia Exchange for, 324
  Multiple Document Interface, 20–32
  resources, 323
  server models, 7
    ASP VBScript server model, 7
    list of supported server models, 8
  specifying scripting language, 46
  toolbars, 32–38
  workspace layout, 19
Dreamweaver 8
  benefits for ASP developer, 9
  news about, 323
DSN (data source name)
  connecting Dreamweaver to blog database, 254
  connecting Dreamweaver to image gallery database, 295
  creating database connections, 113–116
    Access database, 114
    connection with Dreamweaver, 115
    SQL Server database, 114
dynamic ASP
  Application panel, 16
  Code Block icon creating, 13
  Insert bar, 11–16
dynamic ASP pages
  creating simple, 4
  database-driven data exchange, 6
  flow of data with, 6
  myphotos_main.asp, 298–301
  onews_admin_archives.asp, 269
  onews_admin_details.asp, 275
  onews_archives.asp, 265
  onews_details.asp, 263
  onews_main.asp, 260
  web browser view, 5
Dynamic button, Property inspector, 206
Dynamic Data dialog
  myphotos_main.asp, 299
  onews_main.asp, 259
  selecting, 184
  selecting column, 212
Dynamic Data icon
  Application tab, Insert bar, 14
Dynamic List/Menu dialog
  adding code to specify selection, 212
  binding dynamic select lists with selected item, 211
  building Author drop-down menu, 206
  building Category drop-down menu, 207
  building dynamic select lists, 206
dynamic select lists
  binding with selected item, 211
  building, 204–207
dynamic websites
  data exchange, 2
  meaning of, 1
dynamic_values_in_email.asp, 160

E

e-mail
  ASP mail components, 146
  creating mailing list, 170–173
  dynamic interaction, 156–167
  formatting with HTML in AspEmail, 153
  formatting with HTML in CDO, 150
  formatting with HTML in JMail, 155
  general requirements for components, 147
  making property values dynamic, 160–167
  sending by hyperlink, 156–158
  sending via form button, 158–160
  sending with AspEmail, 151–154
  sending with CDO, 148–151
  sending with dynamic form field values, 162–163
  sending with dynamic values, 160–161
  sending with JMail, 154–156
  sending with mail components, 148–156
  sending form values to, 143–156
  sending page to friend, 164–167
  setting up SMTP server, 143–146
Edit Diagram dialog
  image gallery web application, 292, 293
Edit Relationships dialog, 99, 198
editquote.asp page
  linking from allquotes.asp, 184, 188
  Update Record server behavior, 186
  updating for authors/categories, 208–214
edit_caption.asp, 317–320
  link from upload.asp, 319
edit_photo_album.asp, 304–306
Else keyword, VBScript, 69, 71
Else, Else If icons
  ASP tab, Insert bar, 13
ElseIf keyword, VBScript, 69
email_frmbutton.asp, 158, 159
email_hyperlink.asp, 156, 158
empty data type, VBScript, 53
enableDelete function, JavaScript
  onews_admin_details.asp, 272
End icon
  ASP tab, Insert bar, 13

Enterprise Manager, SQL Server
  creating SQL Server database, 85, 86
environment variables, ASP, 79–80
  HTTP_REFERER, 79
  QUERY_STRING, 79
  REMOTE_HOST, 79
  SCRIPT_NAME, 80
  SERVER_NAME, 80
  URL, 80
env_variables.asp, 80
equal to operator, VBScript, 64
exclamation point (!)
  code problems, 200
execute_aspemail.asp, 152, 153
execute_cdo.asp, 148, 149, 151
execute_jmail.asp, 154, 155
Expand (to show local and remote sites) icon
  Files panel, 28
Expand All toolbar
  Insert bar, 37
expectedoutput.asp, 69

F

Favorites tab
  Customize Favorite Objects dialog, 32
File Management button
  Document toolbar, 34
Files panel group, 27–29
  Assets panel, 28
  icons explained, 27
  Snippets panel, 28
folder icon illustrated, 184
foreign keys
  setting up relationships, 98
  table relationships, 220
  tblQuotes table, 196
  values held in, 98
forgotten_pwd.asp, 168, 170
form fields
  naming conventions, 162
Form statement, ASP Request, 135
form tag/element
  Action attribute, 134
  adding to page, 177
  attributes, 134
  building dynamic select lists, 204, 205
  creating with post method, 135–141
  get method, 134
  Method attribute, 134
  methods, 134
  post method, 135
  URL parameters name/value syntax, 134
form values
  sending to e-mail, 143–156
format functions, VBScript, 58
forms
  submitting information between pages, 177
  working with, 133
Forms tab, Insert bar, 177
  List/Menu button, 204
forums
  posting questions, 323
From property
  sending e-mail with AspEmail, 152
  sending e-mail with CDO, 149
FromName property, AspEmail, 154
FTP Log panel, 31
functions, VBScript, 58–61
  array functions, 58
  complete list of, websites for, 58
  conversion functions, 58–59
  date functions, 58
  format functions, 58
  math functions, 58
  string functions, 58, 59–61

G

GET
  submitting information between pages, 177
Get file icon, Files panel, 27
get method
  comparison with post method, 135
  form tag, 134
  receiving data from URL parameter, 141
  retrieving URL parameters, 141
  troubleshooting, 134
Getting Started panel
  Access, Microsoft, 84
Go To icon
  Application tab, Insert bar, 15
Grannell, Craig, 322
greater than operator, VBScript, 64
GROUP BY clause, SQL, 111
Guides feature, Design view, 37

H

Hand tool, Design view, 37
Handheld option
  Style Rendering toolbar, 36
help
  posting questions, 323
Host property, AspEmail, 152

HTML
  formatting e-mail with HTML in CDO, 150
  static HTML page, 4
HTMLBody property
  formatting e-mail with HTML in CDO, 150
HTTP header
  post method, 135
HTTP_REFERER
  environment variables, ASP, 79
hyperlinks
  sending e-mail by hyperlink, 156–158

I

Identity data type
  creating primary keys in SQL Server, 96
If...Then...Else statement, VBScript, 69
If...Then...ElseIf statement, VBScript, 69
If statement, VBScript, 68
If, Else If, Else, End icons
  ASP tab, Insert bar, 13
ifstatement.asp, 68
image gallery web application, 287–324
  ASP pages
    addalbum.asp, 301–303
    delete_action.asp, 320–322
    edit_caption.asp, 317–320
    edit_photo_album.asp, 304–306
    myphotos_main.asp, 295–301
    upload.asp, 307–312
    upload_action.asp, 312–316
  connecting Dreamweaver to, 294–295
  creating database for, 289–293
    tbl_photoalbums, 289
    tbl_photos, 290
  creating photo albums, 288, 301–303
  deleting photo albums, 288, 304–306
  deleting photos, 288, 320–322
  displaying photo albums, 288, 295–301
  displaying photos, 288, 312–316
  editing photo albums, 288, 304–306
  editing photo captions, 288, 317–320
  relationship between tbl_photos and tbl_photoalbums, 292
  storing information in database, 288
  uploading photos, 288, 307–316
IN keyword, SQL, 110
Include icon
  ASP tab, Insert bar, 12
includes, 12
Input Tag Accessibility Attributes dialog, 178, 205
  adding authors and categories to edit page, 210
  building dynamic select lists, 204, 205
  building login form, 225
Insert (author) page illustrated, 201
Insert (category) page illustrated, 202
Insert (quote) page illustrated, 189
Insert bar, 11–16
  Application tab, 14–16
  ASP tab, 12–14
  Coding toolbar, 36
  Document toolbar, 33
  Expand All toolbar, 37
  Forms tab, 177
  List/Menu icon, 204
  Show as Menu view, 11
  Show as Tabs view, 11, 12
  Standard toolbar, 35
  Style Rendering toolbar, 35
  styles for, 11
Insert panel group, 32–37
Insert Record dialog, 180
  addalbum.asp, 302
  After inserting, go to field, 179
  building insertauthor.asp, 200
  inserting categories, 202
  registering new users, 234
  updated to add author/category, 208
Insert Record server behavior, 179–182
  adding data to database, 177
  conditional code, 180
  forcing page to reload after insertion, 179
  onews_admin_archives.asp, 269
  selecting, 186
  updating, 207
Insert Recordset icon
  Application tab, Insert bar, 15
INSERT statement, SQL, 124–125
insert.asp page, 177
  adding authors and categories to edit page, 210
  adding conditional code, 180
  basic Quote submission form, 179
  Insert Record server behavior, 179
  saving to web site, 177
  updating for authors/categories, 203–208
insertauthor.asp page, 199
insertcategory.asp page, 201
int data type
  creating tables in SQL Server, 96
integer data type, VBScript, 53
Internet User (IUSR) account, 89

IP addresses
  configuring SMTP server, 145
  finding your IP address, 152
ipconfig command, 152
IUSR account, 89

J

JavaScript
  scripting ASP in, 46
JavaScript server model, ASP
  getting support for, 8
JMail
  formatting e-mail with HTML in, 155
  properties and methods
    AddRecipient property, 155
    AddRecipientBcc property, 156
    AddRecipientCc method, 155
    ContentType property, 155
    Sender property, 155
    ServerAddress property, 155
  sending cc and bcc with, 155
  sending e-mail with, 154–156
JOIN keyword, 216
joins, database tables
  random quote selector, 214

K

keywords
  database naming conventions, 91

L

Left function, VBScript, 60
Len function, VBScript, 59
len.asp, 59
less than operator, VBScript, 64
Link Checker panel
  Results panel group, 31
Link field, Property inspector illustrated, 191
list element
  creating form elements with post method, 137–139
List/Menu icon, Insert bar
  building dynamic select lists, 204
Live Data View button
  Document toolbar, 34
Local Info screen
  Site Definition dialog, 39
Local view, Files panel, 27
locking
  SQL Server/Access compared, 87, 88
Log In User dialog, 228
Log In User server behavior
  adding, 227–229
  blog security, 278
  expanding code, 237–238
Log Out User server behavior
  adding, 231–232
  blog security, 279
logical operators, VBScript, 61, 62–64
  AND operator, 62
  NOT operator, 63
  OR operator, 62
  precedence, 67
Login Properties dialog box
  SQL Server database, 252
login system
  blog application, 276–280
  building login form, 225–227
  checking new username, 235
  cookies, 241
  creating, 225–232
  creating user-friendly login, 236–237
  Log In User server behavior, 227–229, 237–238
  Log Out User server behavior, 231–232
  redirecting failed logins, 227
  registering new users, 233–236
  “remember me” feature, 239–242
  restricting access, 229–231
login.asp
  blog security, 276, 277–279
  building login form, 225
login_create_cookies.asp, 74
login_create_sessions.asp, 77, 78
login_user.asp, 136
logout.asp, 278–280
long data type, VBScript, 53
looping logic, VBScript, 71–72
  Do...Loop statement, 71
  While...Wend statement, 72
LTrim function, VBScript, 60

M

Macromedia Exchange for Dreamweaver, 324
Magnification list, Design view, 37
mail components
  ASP mail components, 146
  general requirements for, 147
  sending e-mail with, 148–156

mailing lists
  creating, 170–173
mailing_list.asp, 170, 172, 173
Manage Sites dialog, 27
  showing newly created site, 42
many-to-many relationship, 198
Map view, Files panel, 27
Master Detail Page Set icon
  Application tab, Insert bar, 15
math functions, VBScript, 58
mathematical operators, VBScript, 61, 65–66
  precedence, 66
MDI (Multiple Document Interface), 20–32
  Application panel group, 22–25
  CSS panel, 21
  Files panel group, 27–29
  Results panel group, 30–32
  Tag Inspector panel group, 25–27
menu element
  conditional logic, 138
  creating form elements with post method, 137–139
Method attribute, form tag, 134
methods
  form tag, 134
Microsoft Management Console (MMC), 86
MM_redirectLoginSuccess, 237
MM_rsUser recordset, 237, 238
MM_UserAuthorization variable, 228
MM_Username, 228
modulus operator, VBScript, 65
Multiple Document Interface
  see MDI
myMail.Bcc property
  sending cc and bcc with CDO, 151
myMail.Cc property
  sending cc and bcc with CDO, 151
myphotos_main.asp, 295–301
  implementing dynamic features, 298–301
my_first_cookie.asp, 73, 75
my_name.asp, 54

N

naming conventions, 205
  databases, 91
  form fields, 162
  prefixing session variables, 238
  SQL Server objects, 103
  variables, VBScript, 56
navigation
  Recordset navigation status icon, 15
New File panel
  Microsoft Access, 84
NOT keyword, SQL, 112
NOT operator, VBScript, 63
null
  description of, 96
null data type, VBScript, 53
Number data type
  subtypes of, 94

O

O News blog application, 246–284
  see also blogs
ODBC/OLE DB connection strings
  Dreamweaver to blog database, 253
  Dreamweaver to image gallery database, 294
one-to-many relationship, 198
  creating relationships in Access, 99
onews_admin_archives.asp, 266–270, 276, 280
onews_admin_details.asp, 270–276
onews_archives.asp, 264–266, 281
onews_details.asp, 261–263
onews_main.asp, 255–260
onews_searchresults.asp, 281–284
online diary
  see blogs
operator precedence, VBScript, 66–67
  parentheses, 67
operators, VBScript, 61–67
  addition operator, 55
  assignment operators, 61, 62
  comparison operators, 61, 64–65
  concatenation operator, 54
  logical operators, 61, 62–64
  mathematical operators, 61, 65–66
  operator precedence, 66–67
    parentheses, 67
operator_order.asp, 67
OR keyword, SQL, 105
OR operator, VBScript, 62
ORDER BY clause, SQL, 109
  ASC/DESC keywords, 110
output
  printing VBScript variables, 50
Output icon
  ASP tab, Insert bar, 13

P

pages
  restricting access to, 223–243
  submitting information between pages, 177
paging
  Recordset Paging icon, 15
panel group menu option
  pasting recordset using, 209
Panel Group Options menu icon, 21
panel groupings
  Rename Panel Group dialog, 20
  restrictions on, 21
panel groups
  Application panel group, 22–25
  Files panel group, 27–29
  Insert panel group, 32–37
  Results panel group, 30–32
  Tag Inspector panel group, 25–27
Parameters dialog
  illustrated, 185
  myphotos_main.asp, 299
  onews_main.asp, 259
  onews_searchresults.asp, 283
  upload.asp, 310, 311
passwords
  sending user forgotten password, 167–170
Persits Software
  sending e-mail with AspEmail, 151
Personal Web Server (PWS), 143
photo gallery
  see image gallery web application
PI *see* Property inspector
Point to File icon illustrated, 188
POST
  submitting information between pages, 177
post method
  comparison with get method, 135
  creating form elements with, 135–141
    check boxes, 139–141
    list/menu element, 137–139
    text boxes, 136–137
  form tag, 135
  HTTP header, 135
  Request.Form statement, 135
prefixes
  VBScript variable naming conventions, 56
Preview/Debug in Browser button
  Document toolbar, 34
primary keys
  creating in Access, 94
  creating in SQL Server, 95
  foreign keys and, 98
  table relationships, 220
  use of, 92
Print option
  Style Rendering toolbar, 36
programming languages
  case sensitivity, 48
Projection option
  Style Rendering toolbar, 36
Properties panel
  upload.asp, 308
Property inspector (PI), 29
  Dynamic button, 206
  recordset properties, 217
  Results panel group, 30–32
public user role, 90
Put file icon, Files panel, 27
PWS (Personal Web Server), 143

Q

queries
  Access, 102
Query Analyzer, 85
Query designer window, 215
QueryString statement, ASP Request
  retrieving URL parameters with, 141–143
querystrings, 13, 141
QUERY_STRING
  environment variables, ASP, 79
Quote text field properties, 178
quotes
  www.brainyquote.com, 180
Quotes database table
  see tblQuotes table
Quotes recordset
  building, 217–220
Quotes web page
  building allquotes.asp page, 182–185
  building delete.asp page, 189–192
  building editquote.asp page, 186–189
  building insert.asp page, 177–182
  building insertauthor.asp, 199
  building insertcategory.asp, 201
  creating tblQuotes database table, 176
  displaying random quote, 220
  illustrated, 185
  illustrated with Delete links, 191
  random quote selector, 214–220
  updating editquote.asp page, 208–214
  updating insert.asp page, 203–208

R

random quote selector, 214–220
Randomize function, 219
randomquote.asp page, 216
  bindings shown, 218
Record Update dialog
  edit_caption.asp, 318
  edit_photo_album.asp, 305
records, database
  see database records
Recordset builder
  Advanced Recordset builder dialog, 121
  building SQL statements, 116–122
  SQL Server views/Access queries, 102, 103
Recordset dialog
  building Quotes recordset, 217
  building rsCategories recordset, 204
  edit_caption.asp, 317
  edit_photo_album.asp, 304
  illustrated, 182, 187
  myphotos_main.asp, 296
  onews_admin_archives.asp, 267
  onews_admin_details.asp, 271
  onews_archives.asp, 264
  onews_details.asp, 261
  onews_main.asp, 255
  Repeat Region server behavior, 182
  retrieving authors, 203
  searching blogs, 282
  updating rsQuotes recordset, 211
  upload.asp, 307
Recordset icon
  Application tab, Insert bar, 14
Recordset navigation status icon
  Application tab, Insert bar, 15
Recordset Paging icon
  Application tab, Insert bar, 15
recordsets
  building Authors recordset, 203
  building Categories recordset, 204
  building Quotes recordset, 217–220
  description, 14
  MM_rsUser recordset, 237
  recordset properties, 217
  updating editquote.asp page, 208
  updating rsQuotes recordset, 211
redirect code
  delete_action.asp, 322
  upload_action.asp, 315
Reference panel, 20, 32
referential integrity
  tblQuotes/tblAuthors/tblCategories, 198
Refresh Design View button
  Document toolbar, 34
Refresh icon, Files panel, 27
regions
  Repeated Region icon, 14
  Show Region icon, 15
relational databases, 97–103
  relationships in Access, 98–100
  relationships in SQL Server, 100–102
relationships
  creating join in database, 214
  Edit Relationships dialog box, 99, 198
  effect of moving tables in dialog window, 199
  many-to-many relationship, 198
  one-to-many relationship, 198
  Query designer window, 215
  setting up, 97, 98
  tblQuotes/tblAuthors/tblCategories, 198
  tbl_photos/tbl_photoalbums, 292
Relationships dialog, 199
Relationships icon, 197
Relationships toolbar button, 98
relay restrictions options
  configuring SMTP server, 145
“remember me” login feature, 239–242
Remote Info screen
  Site Definition dialog, 40
Remote view, Files panel, 27
REMOTE_HOST
  environment variables, ASP, 79
Remove statement, ASP Session.Contents, 78
Rename Panel Group dialog, 20
Repeat Region dialog
  illustrated, 183
  myphotos_main.asp, 300
Repeat Region server behavior, 182–185
  onews_main.asp, 258
repeated region, 14
Repeated Region icon
  Application tab, Insert bar, 14
Request.Cookies statement, ASP, 73
Request.Form statement, ASP, 135
  retrieving values of form elements, 137
Request.QueryString statement, ASP
  retrieving URL parameters with, 141–143
Request.ServerVariables statement, ASP
  retrieving environment variables, 80
reserved words
  database naming conventions, 91
Response.Cookies statement, ASP, 73
Response.Write code
  printing VBScript variables, 50
Response.Write icon
  ASP tab, Insert bar, 13

Restrict Access To Page dialog, 230
  blog security, 276
Restrict Access To Page server behavior, 231
  creating user-friendly login, 236
Results panel group, 30–32
  FTP Log panel, 31
  Link Checker panel, 31
  Reference panel, 20, 32
  Search panel, 30
  Server Debug panel, 32
  Site Reports panel, 31
  Target Browser Check panel, 31
  Validation panel, 30
rewrite_cookie.asp, 75
rewrite_session.asp, 78
roles
  SQL Server for Internet use, 89
rsAuthors recordset, 203
rsCategories recordset, 204
rsQuotes recordset, 183
  building Quotes recordset, 217–220
  updating to retrieve specific quote, 211
RTrim function, VBScript, 60

S

Save As context menu option
  creating insertauthor.asp, 199
Screen option
  Style Rendering toolbar, 36
SCRIPT_NAME
  environment variables, ASP, 80
Search panel, 30
searching
  blogs, 281–284
security
  blogs, 276–280
  creating login system, 225–232
  creating tblAdmin table, 224
  restricting access to pages, 223–243
  URL parameters, 134
select lists, dynamic
  binding with selected item, 211
  building, 204–207
SELECT statements, SQL, 103–106
  AND keyword, 105
  OR keyword, 105
  WHERE keyword, 104
Send method
  sending e-mail with CDO, 149
Sender property, JMail, 155
sendnameback.asp, 51
send_email_form.asp, 162
send_note.asp, 139, 140
send_page_form.asp, 164, 167
server behaviors
  Delete Record, 189–192
  Insert Record, 179–182, 207
  Log In User, 227–229, 237–238
  Log Out User, 231–232
  Repeat Region, 182–185
  Restrict Access To Page, 231
  Update Record, 186–189, 213
  using commands instead of, 122
Server Behaviors panel, 16, 24
Server Debug panel, 32
server models, 7
Server Side Includes (SSI), 12
Server Variables icon
  ASP tab, Insert bar, 12
Server.CreateObject icon
  ASP tab, Insert bar, 13
ServerAddress property, JMail, 155
ServerVariables statement, ASP Request
  retrieving environment variables, 80
SERVER_NAME
  environment variables, ASP, 80
session object, ASP, 77
session variables, 76–79
  blog security, 278
  creating, 77
  deleting, 78
  prefixing, 238
  retrieving, 77
  server memory, 79
  setting session timeout interval, 77
  timeout scope, 79
  updating, 78
  words of warning for using, 79
Session.Abandon statement, ASP
  deleting session variables, 79
Session.Contents.Remove statement, ASP
  deleting session variables, 78
Session.Timeout statement, ASP, 77
sex_selection.asp, 137
Show as Menu view, Insert bar, 11
Show as Tabs view, Insert bar, 11
Show Region behavior
  onews_main.asp, 258
Show Region dialog
  myphotos_main.asp, 300
Show Region icon
  Application tab, Insert bar, 15
Show Table dialog, 197
  adding tables to query design, 214
  creating relationships, 197
single data type, VBScript, 53

Site Definition dialog
creating new site, 42
Local Info screen, 39
Remote Info screen, 40
Testing Server screen, 41
Site Reports panel, 31
SMTP (Simple Mail Transfer Protocol) server
configuring, 143
relaying e-mail messages, 145
setting up, 143–146
Snippets panel, 28
Split view, 10
Document toolbar and, 34
Dreamweaver page in, 11
SQL (Structured Query Language)
AND keyword, 105
BETWEEN keyword, 108
COUNT keyword, 107
DELETE statement, 128–130
description, 14
DISTINCT keyword, 112
GROUP BY clause, 111
handling dates, 113
IN keyword, 110
INSERT statement, 124–125
manipulating database records, 122–130
NOT keyword, 112
OR keyword, 105
ORDER BY clause, 109
ASC/DESC keywords, 110
Recordset builder, 116–122
SELECT statements, 103–106
SUM keyword, 107
TOP keyword, 108
UPDATE statement, 125–128
WHERE keyword, 104
SQL Server database
Access compared to, 87
creating relationships in, 100–102
creating SQL Server database, 85–88
creating tables, 95–97
database connection with Dreamweaver, 116
Database Properties window, 248
handling dates, 113
Login Properties dialog box, 252
naming conventions, 91, 103
preparing for Internet use, 88–91
setting up DSN to, 114
view of console root, 248
views, 102
SQL Server Service Manager, 247
SQL Server table structure
Dreamweaver table compared to, 176
SSI (Server Side Includes), 12
Standard toolbar, 35
static HTML page, 4
description of limitation, 6
flow of data with, 6
Static options
binding dynamic select lists with selected item, 211
building dynamic select lists, 206, 207
stored procedures, SQL
manipulating database records, 122
string data type, VBScript, 53
concatenation, 54
quotation marks, 53
string functions, VBScript, 58, 59–61
Left function, 60
Len function, 59
LTrim function, 60
RTrim function, 60
Trim function, 60
Style Rendering toolbar, 35
options, 36
Subject property
sending e-mail with CDO, 149
SUM keyword, SQL, 107

T

Tab Index field
Input Tag Accessibility Attributes dialog, 204, 205
table relationships
primary/foreign keys, 220
Query designer window, 215
tables
blog database tables, 249–253
creating, 92–97
Access, 93–95
SQL Server, 95–97
database naming conventions, 91
primary keys, 92
Show Table dialog box, 197, 214
Tag Chooser, 183
Tag Inspector panel group, 25–27
Attributes panel, 26
Behaviors panel, 26
Target Browser Check panel, 31
tblAdmin table, 224
tblAuthors table, 197
creating joins in database, 216
creating relationship with tblQuotes, 197, 198
tblCategories table, 197
creating relationship with tblQuotes, 197, 198
tblQuotes table, 176, 196
basic Quote submission form, 179
building form for adding quotes, 177

creating relationship with tblAuthors, 197, 198
creating relationship with tblCategories, 197, 198
Delete Record server behavior, 189
Dreamweaver table structure, 176
enforcing referential integrity for, 198
foreign keys, 196
Insert Record server behavior, 179
Repeat Region server behavior, 182
rsQuotes recordset, 183
SQL Server table structure, 176
Update Record server behavior, 186
updating, 196
tbl_onews table
adding data to, 251
creating blog database, 249
setting permissions for, 251
tbl_photoalbums
image gallery database, 289
relationship with tbl_photos, 292
tbl_photos
image gallery database, 290
relationship with tbl_photoalbums, 292
tbl_users table
adding data to, 251
creating blog database, 250
setting permissions for, 251
Testing server
Files panel, 27
Testing Server screen
Site Definition dialog, 41
text boxes
creating form elements with post method, 136–137
TextBody property
formatting e-mail with HTML in CDO, 150
sending e-mail with CDO, 149
time data type, VBScript, 53
Timeout statement, ASP Session, 77
timeouts
blog security, 280
To property
sending e-mail with CDO, 149
Toggle CSS option
Style Rendering toolbar, 36
toolbars, 32–38
Coding toolbar, 36–37
Document toolbar, 33–35
Insert toolbar/panel group, 32–37
Standard toolbar, 35
Style Rendering toolbar, 35
TOP keyword, SQL, 108
transaction log files
SQL Server/Access compared, 87
transformations
XSL Transformation icon, 16
Trim function, VBScript, 60
Trimmed Form Element icon
ASP tab, Insert bar, 13
Trimmed QueryString Element icon
ASP tab, Insert bar, 13
troubleshooting
avoiding Dreamweaver bug, 177
get method, 134
TTY option
Style Rendering toolbar, 36
TV option
Style Rendering toolbar, 36
TypeName function, 59

U

Update Record dialog
illustrated, 188
updated, 213
Update Record icon
Application tab, Insert bar, 15
Update Record server behavior, 186–189
onews_admin_details.asp, 273
updating, 213
UPDATE statement, SQL, 125–128
upload.asp, 307–312
edit_caption.asp link, 319
upload_action.asp, 312–316
URL environment variable, ASP, 80
URL parameters
description, 141
name/value syntax, 134, 141
receiving data from, 141
retrieving with Request.QueryString, 141–143
security, 134
User Authentication icon
Application tab, Insert bar, 16
users
creating login system, 225–232
creating user-friendly login, 236–237
expanding Log In User server behavior code, 237–238
registering new users, 233–236
checking new username, 235
“remember me” login feature, 239–242
sending user forgotten password, 167–170
Users object
preparing SQL Server database for Internet use, 89
user_rating.asp, 69, 70

V

Validate Markup button
  Document toolbar, 34
validation
  creating login system, 225–232
Validation panel, 30
varchar data type, 96
variable types, VBScript, 53
variables
  ASP requesting server variables, 12
variables, VBScript, 48–57
  addition operator, 55
  concatenation, 54
  data types, 52
  declaring, 49
    implicitly, 52
    printing variables, 51
    syntax rules, 52
  naming conventions, 56
    meaningful names, 56
    prefixes, 56
  printing, 50
    ASP shortcut (<%=), 50
    declaring variables, 51
VBScript
  ASP VBScript server model, 7
  case sensitivity, 47, 48
  commenting code, 57
  conditional logic, 68–71
    If, Else If, Else, End icons, 13
  cookies, ASP, 72–76
  defining ASP VBScript site in Dreamweaver, 39–42
  functions, 58–61
  looping logic, 71–72
  operators, 61–67
  session variables, ASP, 76–79
  specifying scripting language, 46
  variables, 48–57
View Options button
  Document toolbar, 34
view prefix explained, 216
views, SQL Server, 102
Visual Aids button
  Document toolbar, 35

W

web browsers
  see browser view
web log
  see blogs
web pages
  see pages
web sites
  restricting access to pages, 223–243
weblog
  see blogs
welcome_user.asp, 136, 137
WHERE keyword, SQL, 104
While...Wend statement, VBScript, 72
white space
  Trimmed Form Element icon, 13
workspace layout, 19

X

XSL Transformation icon
  Application tab, Insert bar, 16

Z

Zoom tool
  Design view, 37